Anino BELAN

GRAFIKA V JAZYKU C

(knižnica SDL)

učebný text pre kvartu a kvintu osemročného gymnázia

BRATISLAVA
2012

Copyright © 2012, Anino Belan

Dielo je zverejnené pod licenciou Creative Commons Attribution-NonCommercial-ShareAlike License
http://creativecommons.org/licenses/by-nc-sa/3.0

Obsah

Úvod

Práve ste sa začítali do pokračovania kurzu jazyka C. Toto pokračovanie je venované knižnici SDL (skratka zo Simple DirectMedia Layer). Knižnica SDL je určená na programovanie hier. Programy, ktoré s jej pomocou napíšete, sa dajú skompilovať a spúšťať na množstve operačných systémov (ako príklad uveďme Android, AmigaOS, AmigaOS 4, BeOS/Haiku, iOS, Linux, Mac OS 9, Mac OS X, MorphOS, OpenVMS, PlayStation Portable, Syllable, Symbian, webOS alebo Windows).

Knižnicu pôvodne vytvoril Sam Lantinga, keď v roku 1998 pracoval pre firmu Loki Software. Pracoval totiž na porte hry Doom pre Macintosh a situácia si vyžadovala knižnicu, s ktorej pomocou by sa dala robiť grafika rovnakým spôsobom na rôznych operačných systémoch. Dnes je knižnica dostupná pod licenciou LGPL[1]. Sam je dodnes hlavným koordinátorom a na jej vytváraní sa podieľajú stovky ľudí z celého sveta. Jej funkcie vám dávajú možnosť pristupovať ku grafike, využívať vo vašich programoch zvukovú kartu, pracovať s myšou, klávesnicou a joystickom a programovať, čo sa vám páči.

Táto knižka nadväzuje na Kurz jazyka C. Bolo by dobré, aby ste skôr, než sa do nej pustíte, zvládli prvý diel, alebo sa naučili základy jazyka C nejako inak.

Prajem príjemnú zábavu.

Anino

1 http://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License

1. lekcia

Začíname s grafikou alebo „Už zase Hello, world!“

Jednou z najsilnejších čŕt jazyka C je jeho univerzálnosť. Môžete ho použiť pri tvorbe počítačovej hry, zložitého databázového systému, programu na modelovanie procesov v jadrovom reaktore alebo textového procesora. Nech programujeme čokoľvek, stále používame základné štruktúry C-čka ako napríklad `if`, `while`, `for` alebo `case`.

Je ale zrejmé, že na písanie rôznych druhov programov je treba mať k dispozícii rôzne prostriedky. Keď programujete internetový server, budete používať úplne iné funkcie, než keď programujete hru. A práve na to slúžia jazyku C knižnice. Knižnice obsahujú funkcie z jednotlivých oblastí a programátor môže použiť tie, ktoré potrebuje.

Knižníc pre prácu s grafikou je viacero. Niektoré sú jednoduchšie, niektoré zložitejšie, niektoré sú určené pre konkrétny operačný systém, niektoré sú schopné pracovať pod rôznymi systémami. V nasledujúcich lekciách sa budeme zaoberať knižnicou SDL. Vybrali sme ju preto, lebo je jednoduchá (áno, v porovnaní s inými je skutočne jednoduchá), lebo môže fungovať pod všetkými operačnými a podobnými systémami, na ktoré si bežne spomeniete, pretože sa smie používať zadarmo a pretože ju v súčasnosti pri tvorbe nezávislých hier používa skoro každý.

Prvý program, ktorý v knižnici SDL napíšeme, bude tradičné „Hello, world!“. Na rozdiel od konzolovej verzie tohto programu, v grafickom režime to nie je to najjednoduchšie, čo sa dá programovať. Pri jeho písaní sa dotkneme viacerých tém, ktoré v neskorších lekciách rozoberieme podrobnejšie. Ale keď sa stretnete s trochu zložitejším programom hneď na začiatku, budete mať aspoň akú-takú predstavu, čo to celé má robiť.

Predtým, než sa pustíme do programovania, budeme si ale musieť zohnať nejaké materiály. V našom prvom prípade to bude písmo, ktoré budeme používať. Keď ste pracovali v termináli, váš program typ písma nastaviť nevedel. V grafickom režime máte možnosť písmo zvoliť, ale na druhú stranu, ak chcete písať, nejaké písmo zvoliť musíte. Na internete sa dá nájsť mnoho zaujímavých fontov[2]. V našom príklade budeme používať font Plasma Drip, ktorý si môžete stiahnuť na adrese http://www.ceskefonty.cz/ceske-fonty/plasma-drip. Stiahnite si archív s fontom a rozbaľte. Uistite sa, že sa súbor `plasdrip.ttf` bude nachádzať v tom istom adresári, ako skompilovaný program.

Oproti doterajším zvyklostiam sme pristúpili k jednej zmene: budeme číslovať riadky v programe. Slúži to len na orientáciu v programe (programy budú dlhšie, než doteraz), tak to **nepíšte** do zdrojákov.

2 Napríklad na stránke http://www.ceskefonty.cz/ alebo http://www.1001freefonts.com/

Náš prvý program, ktorý používa knižnicu SDL bude teda vyzerať takto:

```
 1  #include <SDL/SDL.h>
 2  #include <SDL/SDL_ttf.h>
 3
 4
 5  int main(int argc, char *argv[])
 6  {
 7    SDL_Surface *screen = NULL;
 8
 9    SDL_Init( SDL_INIT_EVERYTHING );
10    screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE );
11    TTF_Init();
12    SDL_WM_SetCaption( "Hello, World!", NULL );
13
14    SDL_FillRect(screen,NULL,SDL_MapRGB(screen->format,255,255,255));
15
16    TTF_Font *font = NULL;
17    SDL_Surface *sprava = NULL;
18    SDL_Color cierna = { 0, 0, 0 };
19    font = TTF_OpenFont( "plasdrip.ttf", 28 );
20    sprava = TTF_RenderText_Blended( font, "Hello, world!", cierna );
21
22    SDL_Rect offset;
23    offset.x = (screen->w - sprava->w) / 2;
24    offset.y = (screen->h - sprava->h) / 2;
25
26    SDL_BlitSurface(sprava, NULL, screen, &offset);
27
28    SDL_Flip(screen);
29
30    while (1)
31    {
32      SDL_Event event;
33      SDL_WaitEvent(&event);
34      if ((event.type == SDL_QUIT) ||
35                    (event.type == SDL_KEYDOWN &&
36                              event.key.keysym.sym == SDLK_ESCAPE))
37      {
38        SDL_FreeSurface(sprava);
39        TTF_CloseFont(font);
40        TTF_Quit();
41        SDL_Quit();
42        return 0;
43      }
44    }
45  }
```

Program najprv vysvetlíme. Bolo by dobre, aby ste si najprv prečítali, čo ten program vlastne má robiť a až potom sa ho pokúšali kompilovať, pretože ak používate nejakú knižnicu, kompilácia je trochu zložitejšia a ak tú knižnicu ešte nemáte nainštalovanú, čakajú vás ešte starosti s inštaláciou. Ako to treba robiť, sa dočítate hneď po tom, ako si povieme, čo ten program vlastne robí.

Program začína dvoma príkazmi `#include`. Prvý načíta hlavičkový súbor k samotnej knižnici SDL. Druhý načíta hlavičkový súbor k podsystému, ktorý vie pracovať s TrueType fontami. Nie každý program, ktorý napíšete v SDL totiž musí tento podsystém používať a ak ho nepotrebujete, škoda mrhať systémovými prostriedkami.

Niektorí ľudia sa možno budú diviť, načo sú funkcii `main` nejaké zvláštne parametre. Veď doteraz sme sa bez nich pokojne zaobišli. Parametre `argc` a `argv` sú ale celkom šikovná vec. Ukladá sa do nich spôsob, akým bol program volaný. Ak napríklad spravíte program `hello.exe` a spustíte ho z príkazového riadku ako

```
hello.exe Jozko Mrkvicka
```

v premennej `argc` bude počet slov na príkazovom riadku a v poli `argv` budú jednotlivé reťazce. Teda `argv[0]` bude `hello.exe`, `argv[1]` bude `Jozko` a `argv[2]` bude `Mrkvicka`. V našej ukážke to používať nebudeme, ale možno si nejaké zaujímavé využitie dokážete vymyslieť. V každom prípade, tu uvedený spôsob vytvárania funkcie `main` je ten správny (na rozdiel od lajdáckeho `main()`) a keď pod OS Windows použijete knižnicu SDL, tak si skontroluje, či to máte tak, ako to má byť.

Samotný program začína deklaráciou premennej `screen`, ktorá je smerník na `SDL_Surface`. Totiž – celá práca s grafikou väčšinou pozostáva z preklápania obrázkov z jedného miesta v pamäti na druhé. A `SDL_Surface` je presne tá dátová štruktúra, v ktorej si knižnica uchováva obrázky. Tieto štruktúry budeme nazývať **plocha**. Premenná `screen` bude smerník na plochu, do ktorej budeme v našom programe všetko kresliť. Samotnú plochu vyrobíme o chvíľu.

Na riadku 9 príkazom `SDL_Init` celú knižnicu SDL naštartujeme. Ako parameter sme jej zadali hodnotu `SDL_INIT_EVERYTHING`, ktorá hovorí knižnici, že „naštartuj všetko“. Tým „všetko“ sa myslí časový podsystém, audiosystém, videosystém, ovládanie CD ROM a joystick. Použili sme možnosť pre lenivých. Ak v tom chcete mať poriadok a nechcete mrhať systémovými prostriedkami, môžete inicializovať len to, čo potrebujete. Napríklad ak chcete inicializovať iba videosystém a časovač, spravíte to takto[3]:

```
SDL_Init( SDL_INIT_TIMER | SDL_INIT_VIDEO);
```

Na riadku 10 vytvoríme príkazom `SDL_SetVideoMode` plochu, ktorá sa bude zobrazovať v okne nášho programu. Prvé dva parametre sú rozmery plochy. V našom prípade to bude 640×480 pixelov. Ďalší parameter je farebná hĺbka. Hodnota 32 hovorí, že na každú farebnú zložku – červená, zelená, modrá – pripadne 8 bitov a ďalších 8 bitov pripadá na priesvitnosť. To znamená, že v každej zložke môže byť $2^8 = 256$ úrovní a plocha bude vedieť zobraziť $256 \times 256 \times 256 = 16\,777\,216$ farieb, pričom bude vedieť pracovať s 256 úrovňami priesvitnosti[4]. Posledný parameter `SDL_SWSURFACE` hovorí, že plocha sa má vytvoriť v pamäti počítača. Na tomto mieste sa dajú skombinovať viaceré príznaky. Ak napríklad chcete, aby program namiesto v okne bežal v celoobrazovkovom režime, môžete tam napísať toto:

```
screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE | SDL_FULLSCREEN);
```

Ďalší zaujímavý príznak `SDL_RESIZABLE` umožní, aby okno s vašim programom mohlo za behu meniť veľkosť. Ostatné príznaky nájdete v manuáli. Príznaky sa podobne ako pri funkcii `SDL_Init` kombinujú s pomocou zvislej paličky `|`.

3 Ďalšie možnosti nájdete v manuáli. http://wiki.libsdl.org/moin.cgi/ Zoznam všetkých funkcií nájdete v sekcii API By Name

4 To neznamená, že okno bude priesvitné. Všetky ďalšie plochy, ktoré budú v programe vytvorené, ale budú mať rovnaké parametre a pri ich kopírovaní do hlavnej plochy sa už priesvitnosť uplatniť môže.

Ďalšie dva riadky netreba veľmi komentovať. `TTF_Init();` naštartuje subsystém pre prácu s fontami a `SDL_WM_SetCaption( "Hello, World!", NULL );` nastaví nadpis v titulku okna. Rámik okna môže v závislosti na operačnom systéme vyzerať napríklad tak, ako na obrázku 1. Ako druhý parameter pokojne používajte `NULL`[5].

Obrázok 1: Titulok okna

Na riadku 14 sa konečne začne niečo diať. Príkaz `SDL_FillRect(screen,NULL, SDL_MapRGB(screen->format,255,255,255));` nakreslí na určenú plochu obdĺžnik danej farby. Prvý parameter je smerník na plochu, do ktorej budeme kresliť. V našom prípade je to `screen`. Druhý parameter by mal byť smerník na štruktúru `SDL_Rect` s pomocou ktorej sa v knižnici SDL popisujú obdĺžniky. Ak tam však dáte `NULL`, vyplní sa celá plocha. Posledný parameter určuje farbu. Niektoré funkcie používajú na určenie farby štruktúru `SDL_Color`. V nej môžete nastavovať jednotlivé zložky farby priamo. Funkcia `SDL_FillRect` však potrebuje ako vstup číselný kód farby. A ten záleží nie len od zložiek farby, ale môže závisieť aj od ďalších vlastností plochy (napríklad od jej farebnej hĺbky). Tento číselný kód vám zistí funkcia `SDL_MapRGB`. Ako prvý parameter jej dáte formát plochy do ktorej idete kresliť. Formát danej plochy sa zisťuje tak, ako vidíte v kóde. Potom nasledujú jednotlivé farebné zložky farby ako čísla od 0 do 255. V našom prípade sa teda na štrnástom riadku vymaľuje celá plocha na bielo.

V riadkoch 16 až 20 budeme vyrábať nápis. Na riadku 16 deklarujeme smerník na font, na riadku 17 smerník na novú plochu, do ktorej budeme vyrábať nápis a na riadku 18 vytvoríme premennú typu `SDL_Color` a nastavíme jej jednotlivé farebné zložky na nulu, takže farba bude čierna. Font vytvoríme príkazom `font = TTF_OpenFont( "plasdrip.ttf", 28 );` Prvý parameter je meno súboru s fontom. Druhý je veľkosť písma v pixeloch. Keď je font úspešne načítaný, môžeme vytvoriť plochu s nápisom. Patričný príkaz je

```
sprava = TTF_RenderText_Blended( font, "Hello, world!", cierna );
```

Prvý parameter je použitý font, druhý je text, ktorý sa má vytvoriť a tretí je farba textu.

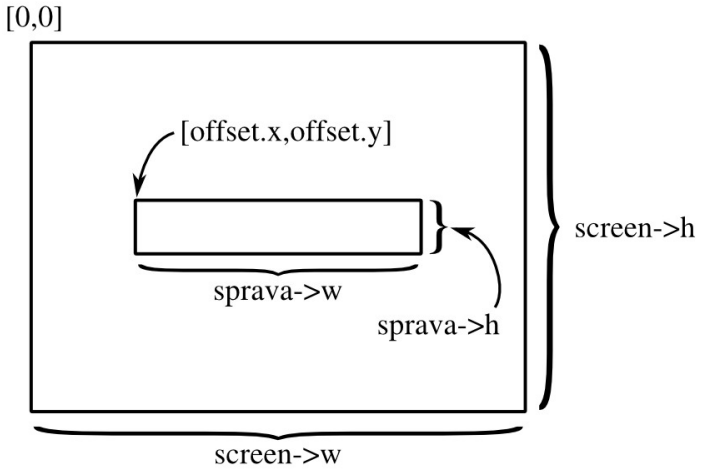
Plochu `sprava`, v ktorej je náš nápis, máme úspešne vytvorenú. Teraz ju treba skopírovať na plochu `screen`. A navyše by sme ešte boli radi, keby sa nám podarilo skopírovať ju presne do stredu.

Na kopírovanie kusu jednej plochy do druhej slúži funkcia `SDL_BlitSurface`, ktorá má štyri parametre. Prvý je smerník na plochu, z ktorej sa bude kopírovať. V našom prípade to teda bude `sprava`. Druhý parameter je smerník na štruktúru typu `SDL_Rect`, do ktorého sa dá uložiť obdĺžnik a ktorý popisuje, ktorý konkrétne kus zdrojovej plochy sa má kopírovať. Ak sa kopíruje celá plocha, ako je to v našom prípade, stačí tam dať `NULL`. Tretí parameter je smerník na plochu, do ktorej budeme kopírovať – v našom prípade `screen`. Štvrtý parameter bude zase smerník na `SDL_Rect`. Tento bude určovať, kam sa má obrázok skopírovať. Štruktúra `SDL_Rect` má štyri zložky: `x` a `y` sú súradnice ľavého horného rohu, `w` a `h` sú šírka a výška obdĺžnika. V prípade štvrtého parametra sa ale druhé dve zložky ignorujú (rozmery kopírovanej časti určuje druhý parameter) a funkciu `SDL_BlitSurface` zaujímajú iba zložky `x` a `y`.

Aby sme teda plochu `sprava` skopírovali presne do stredu obrazovky, musíme si vytvoriť štruktúru `SDL_Rect` a do jej zložiek `x` a `y` nastaviť súradnice ľavého horného rohu miesta, kam sa má naša správa umiestniť. To sa deje na riadkoch 22 až 24. Štruktúru sme si nazvali `offset`. Súradnicu `x` ľavého horného rohu vypočítame tak, že zistíme rozdiel medzi šírkou obrazovky a šírkou správy a potom ho vydelíme dvoma, pretože polovica toho rozdielu musí byť pred správou

5 Tých, čo chcú vedieť, na čo je druhý parameter dobrý, opäť odkazujeme na manuál.

a polovica za správou. (Dobre si pozrite na obrázku 2, ako to funguje.) Podobne súradnica `y` ľavého horného rohu bude polovica rozdielu výšky obrazovky a výšky správy.



Obrázok 2: Umiestnenie správy

Pre niekoho môže byť mätúce, že bod so súradicami [0,0] sa nachádza v ľavom hornom rohu obrazovky a súradnica `y` rastie smerom nadol. Nebojte sa, časom si zvyknete.

Keď máme `offset` správne nastavený, na riadku 26 konečne plochu `sprava` skopírujeme do `screen`.

Do plochy `screen` sme nakreslili všetko, čo sme potrebovali. Teraz potrebujeme zabezpečiť, aby sa to ocitlo aj na monitore počítača. Na to slúži riadok 28 a príkaz `SDL_Flip(screen)`. To, že sa veci neukazujú na monitore hneď, ale až keď sa zavolá `SDL_Flip` má dobrý dôvod. Monitor sa (v závislosti od typu) prekresľuje približne šesťdesiatkrát za sekundu odhora nadol. Ak by sa na ňom objavil nejaký nový objekt v nesprávnom momente, mohlo by sa stať, že sa nakreslí iba jeho dolná časť a vrchná sa nakreslí až pri druhom prekresľovaní obrazovky. Ak sa objekt pohybuje, ľahko sa môže stať, že bude pri takomto prekresľovaní škaredo blikať. A funkcia `SDL_Flip` zabezpečí dve veci. Jednak tú, že sa bude prekresľovať celá stránka naraz a jednak tú, že sa s tým začne presne v tom momente, keď je prekresľovanie na začiatku obrazovky. Obe tieto veci výrazne znižujú blikanie pri animáciách.

Všetko, čo potrebujeme, už na monitore máme. Zostáva už iba počkať, kým sa prípadný používateľ nášho nápisu nabaží a program ukončí a potom po sebe upratať. To majú na starosti riadky 30 až 47. Čakanie na koniec programu bude uzavreté v jednom nekonečnom cykle. Keby tam totiž nič podobné nebolo, program by síce všetko nakreslil, ale skončil by skôr, než by sme si to stihli všimnúť.

Prvý problém bude zistiť, že používateľ už chce skončiť. Akékoľvek zásahy zo strany používateľa spracúva knižnica SDL s pomocou udalostí. Udalosť je pohyb či kliknutie myši, stlačenie klávesy na klávesnici, zmena rozmerov okna alebo jeho zatvorenie a ešte niektoré iné veci.

Na riadku 32 sme si deklarovali premennú `event`, ktorá je typu `SDL_Event` a v ktorej tú udalosť budeme skladovať. Potom sme zavolali funkciu `SDL_WaitEvent`, ktorá čaká, kým nejaká udalosť nastane a keď sa tak stane, vloží jej popis do štruktúry `event`.

Keď sa nám nejakú udalosť podarí zachytiť, musíme sa pozrieť, čo je zač. V položke `event.type` nájdeme jej typ. V prípade, že je typ `SDL_QUIT`, znamená to, že niekto zavrel okno s aplikáciou. V prípade, že je typ `SDL_KEYDOWN`, znamená to, že niekto stlačil klávesu. Náš

program chceme ukončiť buď vtedy, keď niekto okno zavrie, alebo vtedy, keď niekto stlačí klávesu `Escape`. To znamená, že v prípade, že nastala udalosť stlačenia klávesy, musíme sa ešte pozrieť, ktorá klávesa to bola. Musíme sa opäť pozrieť do štruktúry `event`. V položke `event.key.keysym.sym` je uložený kód stlačenej klávesy. Kód klávesy `Escape` je `SDLK_ESCAPE`.

Keď teda vieme, že používateľ chce skončiť používať program, treba po sebe upratať. To majú na starosti riadky 36 až 39. Najprv uvoľníme z pamäti plochu `sprava`. Slúži na to funkcia `SDL_FreeSurface`. Plochu `screen` uvoľňovať nemusíme, o to sa postará samotná knižnica. Potom príkazom `TTF_CloseFont(font)` uvoľníme použité písmo. Príkazom `TTF_Quit()` ukončíme prácu subsystému pre prácu s fontami a príkazom `SDL_Quit()` ukončíme grafický režim a necháme knižnicu SDL ukončiť všetko, čo potrebuje. Príkaz `return 0` potom definitívne ukončí náš program.

Program teda máme napísaný. Teraz ho treba skompilovať. Keď sa o to pokúsite spôsobom známym zo skrípt o jazyku C, pravdepodobne na vás kompilátor vychrlí nejaké chyby. To je spôsobené jednou z troch možných príčin.

Prvá možná príčina je, že ste program odpísali zle, nevymazali ste čísla riadkov, keď ste program kopírovali zo skrípt alebo ste vyviedli niečo podobné a kompilátor by protestoval, aj keby bolo všetko nainštalované tak, ako má byť. Vtedy stačí chybu v kóde opraviť a môžete pokračovať.

Druhá možná príčina je, že nemáte nainštalovanú knižnicu SDL. V tom prípade vám bude kompilátor písať niečo v zmysle

```
hello.c:1:22: fatal error: SDL/SDL.h: Adresár alebo súbor neexistuje
```

prípadne

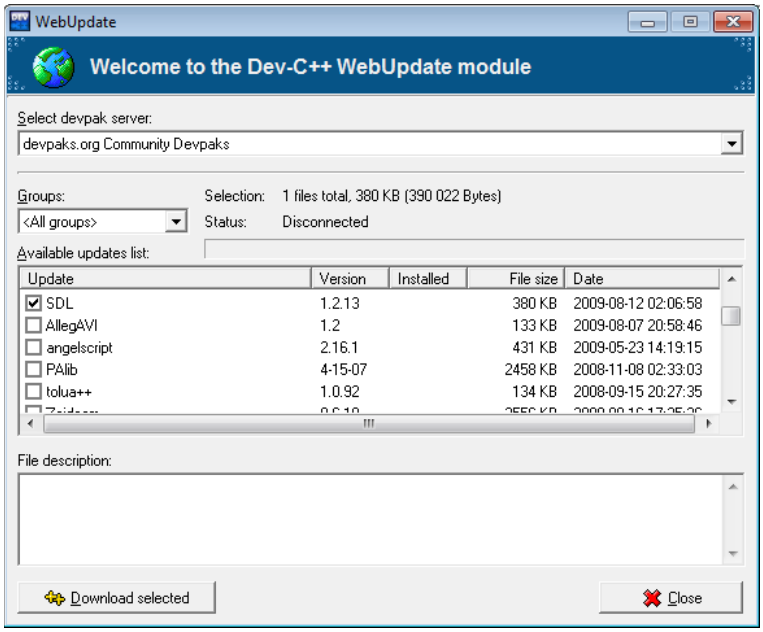
```
SDL/SDL.h: No such file or directory.
```

Z týchto hlášok sa dá vyrozumieť, že to celé padlo na tom, že kompilátor nenašiel súbor `SDL.h`, ktorý sa pokúšame načítať v prvom riadku programu. Znamená to, že knižnicu SDL bude treba doinštalovať.

V prípade, že ste používateľom niektorej distribúcie OS Linux, stačí spustiť váš obľúbený program na správu balíčkov a nájsť ten správny. O jeho stiahnutie a nainštalovanie sa postará systém. Názov balíčku bude začínať na `libSDL` a v názve by malo byť aj slovo `devel`. Je totiž zvykom, že v linuxe sa ku každej knižnici zvyknú robiť dve verzie balíčkov. Tá bez „devel“ je pre tých, ktorí iba chcú používať programy, ktoré knižnicu využívajú. A tá s „devel“ je pre developerov – teda pre programátorov. A práve v nej sa nachádzajú také veci, ako je hlavičkový súbor, ktoré obyčajným používateľom nie sú potrebné, ale bez ktorých si programátor ani neškrtne. A keď už inštalujete, nainštalujte si aj knižnice `libSDL_ttf` (aby ste mohli pracovať s písmami), `libSDL_gfx` (aby ste mohli kresiť geometrické útvary), `libSDL_image` (aby ste mohli načítavať obrázky vo všetkých bežných formátoch), `libSDL_sound` (aby ste mohli používať všetky možné zvukové formáty, napr. mp3), `libSDL_mixer` (aby sa s tými zvukmi dalo pohodlne pracovať) a `libSDL_net` (keby ste náhodou chceli písať niečo, čo sa bude hrať po sieti). Samozrejme zo všetkého nainštalujte tú `devel` verziu.

V prípade, že pracujete pod OS Windows a používate vývojové prostredie Dev-C++, situácia je tiež relatívne jednoduchá. Vojdete do menu `Nástroje` a vyberiete `Vyhľadať aktualizácie/balíčky…` Ukáže sa okno, ktoré môžete vidieť na obrázku 3. V hornom menu s nápisom `Select devpak server:` vyberiete možnosť devpaks.org, pretože tá, ktorá je tam

štandardne, nefunguje a stlačíte tlačidlo `Check for updates`. Správca načíta balíčky, ktoré sú k dispozícii.



Obrázok 3: Správca balíčkov Dev-C++

Zaškrtnite všetky balíčky, ktoré hodláte inštalovať (odporúčame `SDL`, `SDL_ttf`, `SDL_gfx`, `SDL_image`, `SDL_mixer` a `freetype`[6]) a stlačte `Download selected`. Balíčky sa stiahnu a pre každý sa spustí inštalačná rutina. Vy iba klikáte na tlačidlá `Next` a `Finish`.

Program je v poriadku, všetky knižnice sa vám podarilo úspešne nainštalovať, konečne sa dostávame k tretej možnosti, prečo vám kompilácia nezbehne. Opäť sa pozrime na chybové hlášky, ktoré na nás kompilátor vychrlí. Je ich veľa, pod Linuxom vyzerajú približne takto:

```
hello.c:(.text+0x19): undefined reference to `SDL_Init'
hello.c:(.text+0x3d): undefined reference to `SDL_SetVideoMode'
hello.c:(.text+0x46): undefined reference to `TTF_Init'
```

v Dev-C++ takto:

```
[Linker error] undefined reference to `SDL_Init'
[Linker error] undefined reference to `SDL_SetVideoMode'
[Linker error] undefined reference to `TTF_Init'
```

v oboch prípadoch je tam tých riadkov viacero. Tieto chyby hovoria, že prvá fáza kompilácie prebehla úspešne. Kompilátor vyrobil z vášho zdrojového kódu objektový súbor. Všetko sa pokazilo až v druhej fáze kompilácie, v linkovaní. Linkeru totiž nikto nepovedal, kde má funkcie, ktoré používame a ktoré sme mu nasľubovali v hlavičkovom súbore `SDL.h` hľadať. A tak ich všetky ofrfle s komentárom `undefined reference`, po slovensky „nedefinovaný odkaz“.

Takže čo s tým? Treba kompilátoru nejako dohovoriť. Opäť sa venujme najprv OS Linux. Aby linker vedel, aké knižnice okrem štandardných má pripojiť, do príkazu pre linkovanie[7] treba na koniec pridať parametre v tvare `-l<meno knižnice>`. Takže náš program môžete skompilovať príkazom

```
gcc -o hello hello.c -lSDL -lSDL_ttf
```

6 `freetype` je knižnica, ktorú používa ku svojej činnosti `SDL_ttf` a bez nej nefunguje.

7 Alebo ak kompilujete jedným príkazom, tak do toho jedného príkazu.

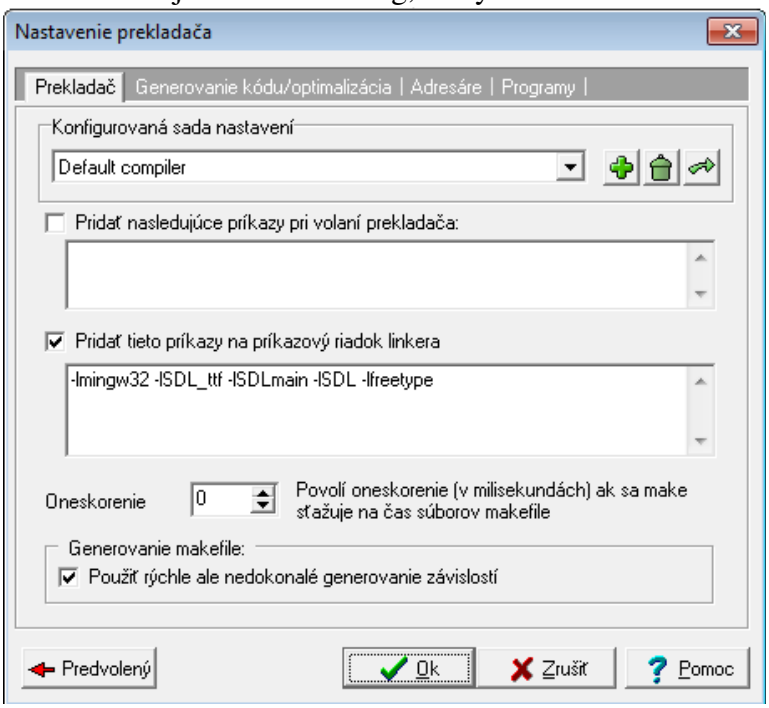
Sú ale ľudia, čo si zvykli používať staré dobré `make`. Keď ale napíšete `make hello`, kompilátor sa spustí bez parametrov, ktoré by mu povedali, aké knižnice treba prilinkovať. Dá sa s tým niečo robiť? Áno, dá.

Chovanie programu `make` sa totiž dá ovplyvniť vytvorením súboru s názvom `Makefile`. V prípade, že budete vytvárať jednoduché miniprojekty, ktoré sa vám celé vojdú do jedného súboru, treba do súboru `Makefile` napísať toto zaklínadlo:

```
%: %.c Makefile
	$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $< -lSDL -lSDL_ttf
```

Je dôležité, aby druhý riadok začínal tabulátorom a nie medzerami a aby na jeho konci boli tie knižnice, ktoré chcete prilinkovať. Momentálne nebudeme vysvetľovať, čo to zaklínadlo znamená[8], keď ale použijete príkaz `make hello` v adresári, v ktorom je súbor `Makefile` s uvedeným obsahom, automaticky sa prilinkujú aj uvedené knižnice.

Ak pracujete pod Windows a používate Dev-C++, choďte do menu `Nástroje` a zvoľte `Nastavenie prekladača`. Objaví sa vám dialóg, ktorý môžete vidieť na obrázku 4.



Obrázok 4: Nastavenie prekladača

Knižníc, ktoré musíte pridať, bude o niečo viacej, ako v linuxe. Do príkazov pre príkazový riadok linkera pridajte `-lmingw32 -lSDL_ttf -lSDLmain -lSDL -lfreetype`.

Nastavenia knižníc si pre svoje projekty musíte spravovať sami. Keď začnete vytvárať nový projekt, nastavenie knižníc ostane rovnaké. Je ale zbytočné linkovať knižnicu SDL k jednoduchým konzolovým programom, takže ju pripájajte iba tam, kde treba.

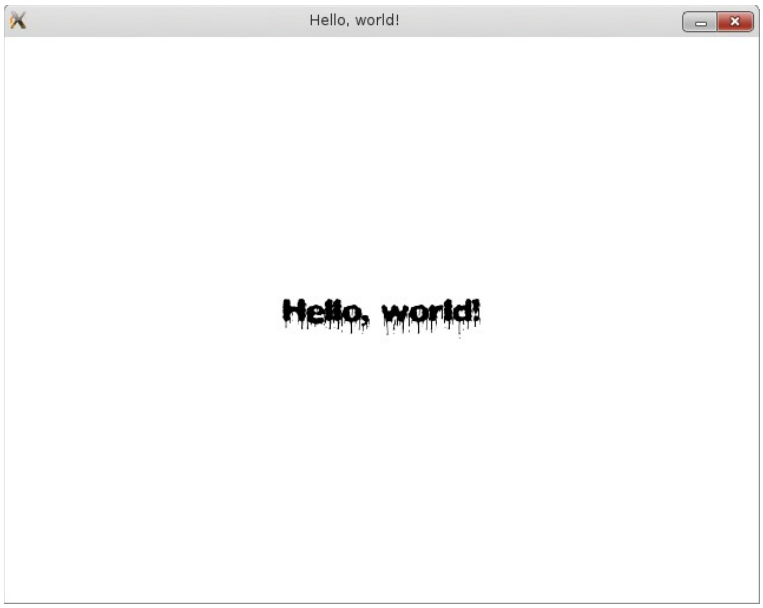
Úloha 1: Nainštalujte si knižnice, rozbehajte si kompilátor a vyskúšajte ukážkový program.

Ak ste úspešne splnili úlohu 1, môžete sa pokochať pohľadom podobným tomu, ktorý môžete vidieť na obrázku 5.

Podaktorí menej šťastnejší z čitateľov vynaložili veľkú námahu. Nainštalovali všetko, čo mali, kompilátor im program skompiloval, ale výsledný program nechce fungovať, vypíše niečo

8 Prípadní záujemcovia nájdu detaily na stránke http://www.gnu.org/software/make/manual/make.html ale pozor, je tam toho naozaj veľa.

v zmysle `Chyba segmentácie` a padne. Tomuto stavu sa dá ľahko odpomôcť a plynie z neho jedno poučenie.



Obrázok 5: Hello, world!

To, že program robí tieto hlúposti je pravdepodobne spôsobené tým, že ste nedali na dobré rady a v adresári s programom nemáte súbor `plasdrip.ttf`, v ktorom je uložený použitý font. Napraví sa to jednoducho. Súbor s fontom dáte tam, kam treba.

To poučenie je ale dôležitejšie: Nepokladajte za samozrejmé, že všetko, čo počítaču prikážete, sa mu aj podarí urobiť. Funkcia `TTF_OpenFont` na riadku 19 má za úlohu font načítať. Ale keď tam ten font nie je, bude sa snažiť márne. V prípade neúspechu preto namiesto smerníku na font vráti hodnotu `NULL`. Preto je vhodné po volaní funkcie `TTF_OpenFont` zistiť, čo vlastne vrátila a zariadiť sa podľa toho.

Takže na začiatok programu treba pridať staré dobré `#include <stdio.h>` a za riadok 19 po načítaní fontu pridajte kód

```
if (font == NULL)
  {
    printf("Nepodarilo sa načítať font\n");
    return 1;
  }
```

Ak sa z nejakého dôvodu nepodarí font načítať, program vám to povie a neskončí s nejakou tajomnou hláškou.

Úloha 2: Nájdite si na internete dokumentáciu ku knižnici `SDL_ttf` (Google je kamarát) a prečítajte si tam všetky detaily k funkcii `TTF_OpenFont`.

Úloha 3: Nájdite v dokumentácii vhodnú náhradu pre funkciu `TTF_RenderText_Blended` a upravte program tak, aby vám to do plochy `sprava` vyrobilo čierny text na červenom pozadí. Zvyšok pozadia v okne zostane biely.

Úloha 4: Upravte program tak, aby sa text zobrazil na vrchu okna. Upravte program tak, aby sa text zobrazoval v pravom dolnom rohu. Spravte to tak, aby to fungovalo, aj keď zmeníte v riadku 10 rozmery okna.

2. lekcia

Farbičky a čiaročky alebo „Výtvarná výchova pre škôlkárov“

Predošlá lekcia bola o tom, ako napísať a spustiť program, ktorý používa knižnicu `SDL`. Popri tom ste sa naučili viacero užitočných vecí. Naučili ste sa, ako kompilátoru vysvetliť, aké knižnice má pri kompilovaní používať, naučili ste sa, ako mazať obrazovku, naučili ste sa nejaké základy okolo používania udalostí a v neposlednom rade ste sa naučili používať knižnicu `SDL_ttf`, ktorá je ku knižnici `SDL` pridružená. Táto a nasledujúca kapitola budú o inej pridruženej knižnici, ktorá sa nazýva `SDL_gfx` a ktorá má na starosti kreslenie grafických primitív.

Povedzme si najprv pár slov o farbách. Ako ste si mohli všimnúť v predošlej kapitole, v tom, ako sa v knižnici `SDL` a pridružených knižniciach popisujú farby vládne taký jemný chaos. Niektoré funkcie používajú štruktúru `SDL_Color`, niektoré chcú mať hodnotu farby vypočítanú s pomocou `SDL_MapRGB` a to v tejto kapitole pribudnú ešte ďalšie dva spôsoby. Všetky tieto prístupy majú ale niečo spoločné. Používajú takzvaný RGB model. To znamená, že každej farbe určíme, aká veľká je jej červená (red – `R`) zložka, aká veľká je zelená (green – `G`) zložka a aká veľká je modrá (blue – `B`) zložka. Z týchto základných zložiek potom môžete namixovať ďalšie farby. Napríklad jasnožltá má červenú a zelenú zložku úplne naplno a modrú úplne stiahnutú, biela sú všetky tri zložky naplno a čierna má všetky tri stiahnuté.[9]

V niektorých situáciách sa k farbe pridáva ešte štvrtá zložka – priesvitnosť (označuje sa alfa – `A`). Ak je alfa nastavená naplno, farbou sa podklad úplne prekreslí. Keď budete alfu znižovať, bude podklad čím ďalej tým viac presvitať a ak ju úplne stiahnete, nebude sa kresliť nič.

Hodnoty `R`, `G`, `B` a `A` sa v prípade knižnice `SDL` zadávajú ako celé čísla od 0 do 255. Niektorí programátori používajú šestnástkovú sústavu a tak namiesto 255 napíšu `0xff`. Jazyku C je jedno, či mu čísla zadávate v desiatkovej, alebo v šestnástkovej sústave. Ale v prípade, že hodláte použiť šestnástkovú, treba pred číslo napísať to `0x`. Šestnástková sústava má na rozdiel od desiatkovej šestnásť cifier: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, `a`, `b`, `c`, `d`, `e` a `f`. Pri zápise teda napríklad `a` bude znamenať 10 a `f` 15. Keď v desiatkovej sústave napíšete 42, znamená to „štyri desiatky a dve jednotky“. Podobne, keď v šestnástkovej sústave napíšete `ff`, znamená to „pätnásť šestnástok a pätnásť jednotiek“ čo je zhodou okolností práve 255.

Výhoda šestnástkovej sústavy sa prejaví, keď treba zapísať naraz celú farbu. (Pozor, nie všetky funkcie takýto zápis umožňujú. V tejto lekcii ale také budú.) Keby sme napríklad chceli zapísať „žltú s päťdesiatpercentnou priesvitnosťou“, dá sa zapísať ako `0xffff0080`. Prvé dve `f` znamenajú „červená naplno“, druhé dve `f` znamenajú „zelená naplno“, ďalšie dve nuly znamenajú „modrú stiahnuť“ a tá `80` na konci znamená „priesvitnosť na polovicu“. Keby sme to číslo zapisovali v desiatkovej sústave, nedalo by sa tak dobre rozoznať, kde končí popis jednej farby a kde začína popis druhej.[10]

Úloha 1: Odhadnite, aká farba bude `R`=224, `G`=27, `B`=76. Odhadnite, aká farba bude `0x4f780cff`.

---

9 Ak chcete vedieť hodnoty R, G a B nejakej inej farby, môžete ich zistiť napríklad na stránke http://www.colorpicker.com

10 Na stránke http://www.colorpicker.com si môžete zistiť skoro celý zápis farby v takomto formáte (konkrétne všetko okrem alfy) v tom rámiku nad výberom farby.

O farbách teda máme akú-takú predstavu, môžeme ísť kresliť. Začneme od najjednoduchšieho, od úsečiek. Na kreslenie úsečiek máme dve funkcie, ktoré sa líšia práve v spôsobe, ako sa im zadáva farba. Ak chceme na ploche `surface` nakresliť úsečku z bodu `[12,28]` do bodu `[621,422]` a chceme, aby bola kreslená farbou `0x4f780cff`, použijeme funkciu

```
lineColor(surface,12,28,621,422,0x4f780cff);
```

Ak chceme nakresliť tú istú úsečku na tú istú plochu a farbu určovať po jednotlivých zložkách `R = 224`, `G = 27`, `B = 76` a `A = 255`, použijeme funkciu

```
lineRGBA(surface,12,28,621,422,224,27,76,255);
```

Použitie funkcie si môžete pozrieť v nasledujúcom programe:

```
 1  #include <SDL/SDL.h>
 2  #include <SDL/SDL_gfxPrimitives.h>
 3
 4  int main(int argc, char *argv[])
 5  {
 6    SDL_Surface *screen = NULL;
 7
 8    SDL_Init( SDL_INIT_EVERYTHING );
 9
10    screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE);
11    SDL_WM_SetCaption( "Ciary", NULL );
12
13    SDL_FillRect(screen,NULL,SDL_MapRGB(screen->format, 0, 0, 0));
14
15    int i;
16    for(i = 0; i < screen->h; i = i + 5)
17      lineColor(screen, 0, 0, screen->w - 1, i, 0x4f780cff);
18
19    SDL_Flip(screen);
20
21    while (1)
22    {
23      SDL_Event event;
24      SDL_WaitEvent(&event);
25      if ((event.type == SDL_QUIT) ||
26            (event.type == SDL_KEYDOWN &&
27                    event.key.keysym.sym == SDLK_ESCAPE))
28      {
29        SDL_Quit();
30        return 0;
31      }
32    }
33  }
```

Až po štrnásty riadok sa deje v podstate to isté, ako v príklade z predošlej lekcie. Drobné rozdiely sú iba v tom, že neštartujeme veci týkajúce sa práce z fontami, na druhom riadku načítame správny hlavičkový súbor, na riadku 11 dáme oknu iný nadpis a na riadku 13 vyplníme okno čiernou namiesto bielej.

Podstatná vec tohto programu sa udeje až na riadkoch 15–17. V cykle tam kreslíme úsečky, ktoré začínajú v bode `[0,0]` a končia na pravej strane okna (x-ová súradnica je

`screen->w - 1)`, pričom y-ová súradnica sa zakaždým zväčšuje o 5 až kým nepríde na spodnú stranu okna.

Riadky 21–32 sú zase podobné ako v prvom programe. Čaká sa buď na udalosť zavretia okna alebo na udalosť stlačenia `Escape`, program po sebe uprace a skončí.

Úloha 2: Pochopte a vyskúšajte. Nezabudnite linkeru pridať parameter `-lSDL_gfx`. Používateľom OS Windows sa môže stať, že sa im súbor `SDL_gfx.dll` nesprávne nainštaloval do adresára `Dev-Cpp\dll` namiesto do `Dev-Cpp\bin`. Ak chcete, aby vám skompilovaný program fungoval, presuňte si súbor `SDL_gfx.dll` do správneho adresára ručne.

Úloha 3: Skúste nahradiť riadok 17 nasledujúcim riadkom:

```
lineRGBA(screen, 0, screen->h - i, screen->w - 1, i,
     224 + (i*31/screen->h),
     27  + (i*228/screen->h),
     76  + (i*179/screen->h),
     255);
```

vyskúšajte a pochopte. Keď sme už raz vybrali základnú farbu `(R:224,G:27,B:76)`, čísla 31, 228 a 179 neboli zvolené len tak náhodne. Odkiaľ sa vzali? Akú farbu bude mať úsečka, ak bude hodnota v premennej `i` presne `screen->h`? Čo to spraví, ak niektoré z čísel 31, 228 alebo 179 zväčšíte?

Úloha 4: Upravte program z úlohy 3, aby sa v okne objavil farebný prechod z modrej do bielej.

Úloha 5: Nakreslite domček so šikmou strechou.

Úloha 6: Nakreslite mriežku 8×8 štvorčekov, každý so stranou 10.

Úloha 7: Ak chcete nakresliť úsečku, ktorá má väčšiu hrúbku, než 1 bod, slúži na to funkcia `thickLineColor`, prípadne funkcia `thickLineRGBA`. Pozrite si v manuáli, ako tie funkcie presne fungujú[11] a použite ich v programe.

Úloha 8: Na kreslenie úsečiek sa dá použiť aj funkcia `aalineRGBA`. Tie dve a-čka na začiatku znamenajú, že sa bude používať antialiasing, teda že funkcia nebude len tak jednoducho zažínať a zhasínať body na ploche, lebo úsečka by bola zubatá, ale pokúsi sa veci vyhladiť. Tento spôsob je pomalší, ale efektnejší. Vyskúšajte ju použiť v ukážkovom programe a porovnajte rozdiel.

11 Manuál nájdete na stránke http://www.ferzkopp.net/joomla/content/view/19/14

3. lekcia

Trojuholníky, štvorčeky a kolieska alebo „Prvácke vystrihovačky“

Úvodný kurz kreslenia máte úspešne za sebou. V tejto lekcii sa budeme venovať pokročilejším témam. Teda asi tak na úrovni výtvarnej výchovy v prvej triede základnej školy. Budeme kresliť obdĺžniky, kolieska, elipsy, troj a viacuholníky. Rovnako, ako v predošlej lekcii budeme okrem knižnice `SDL` používať nadstavbu `SDL_gfx`.

Vzhľadom na to, že ukážkové programy by sa od ukážkového programu z predošlej lekcie líšili iba v detailoch, nebudeme uvádzať celý zdrojový kód. Zoberte zdrojový kód zo strany 15, vynechajte z neho riadky 15 až 19 a na ich miesto napíšte to, čo od vás bude vyžadovať táto lekcia. Okrem toho pridajte na začiatok súboru riadok `#include <time.h>`. V tejto lekcii sa totiž budeme hrať s generátorom pseudonáhodných čísel a aby tie pseudonáhodné čísla boli aspoň trošku náhodné, je zvykom znáhodňovadlo[12] nastavovať podľa aktuálneho systémového času. Tiež môžete aktualizovať nadpis okna (11. riadok v pôvodnom programe).

Takže poďme sa pozrieť na prvú ukážku.

```
1       srand(time(NULL));
2       int i;
3       for(i = 0; i < 10000; i++)
4           boxRGBA(screen,
5               rand() % screen->w, rand() % screen-> h,
6               rand() % screen->w, rand() % screen-> h,
7               rand() % 256, rand() % 256, rand() % 256, 255);
8       SDL_Flip(screen);
```

Na prvom riadku voláme funkciu `srand`, ktorá tam nie je pre srandu králikom, ale na to, aby nastavila znáhodňovadlo. Ako parameter dostane hodnotu `time(NULL)`, čo je počet sekúnd od polnoci 1. januára 1970. Každú sekundu sa tak nastaví iné znáhodňovadlo.

V nasledujúcom cykle, ktorý zbehne desaťtisíckrát sa zavolá funkcia `boxRGB`, ktorá bude kresliť náhodne umiestnené obdĺžniky náhodnej farby. Prvý parameter je smerník na plochu, na ktorú kreslíme. Ďalšie dva parametre (riadok 5) určujú súradnice ľavého horného bodu kresleného obdĺžnika. Súradnicu `x` vyrábame tak, že zavoláme funkciu `rand()`, ktorá nám vyrobí úplne náhodný `int` a potom zistíme jeho zvyšok po delení hodnotou `screen->w`. Ak je šírka plochy 640, zvyšok po delení bude nejaké číslo od 0 do 639, čo je presne to, čo potrebujeme. Rovnako vyrobíme súradnicu `y`, tam ale budeme zisťovať zvyšok náhodného čísla po delení `screen->h`, čo je výška plochy.

12 Znáhodňovadlo je voľný preklad z anglického *random seed*. Je to hodnota, ktorú keby sme nenastavili, náhodné čísla by boli rovnaké pri každom spustení programu (môžete si to vyskúšať). Keď chceme mať zakaždým iné náhodné čísla, treba zakaždým nastaviť iné znáhodňovadlo a systémový čas je na to vhodný, pretože sa stále mení. Generátor pseudonáhodných čísel je dobrý na jednoduchšie veci, ale napríklad na šifrovanie sa nehodí. Prípadní záujemcovia o problematiku si môžu pozrieť krásnu úlohu zo súťaže IPSC a jej riešenie: http://ipsc.ksp.sk/contests/ipsc2007/real/problems/k.php Pozor, je to po anglicky, zaujímavá je tá časť „difficult input data set“.

Ďalšie dva parametre určujú súradnice pravého dolného rohu a vyrobíme ich rovnakým spôsobom.[13]

Na riadku 7 nastavujeme obdĺžniku farbu. Prvé tri parametre majú byť náhodné čísla od 0 do 255. Opäť použijeme tú istú fintu, ako predtým a zistíme si zvyšok náhodného čísla po delení 256. Posledný parameter bude pevne hodnota 255, nech obdĺžniky nie sú priesvitné.

Keď cyklus dobehne, na riadku 8 necháme zobraziť, aký zázrak sme to vytvorili.

Úloha 1: Vyskúšajte.

Úloha 2: Čo sa zmení, keď namiesto uvedeného kódu použijete nasledujúci kód?

```
 1       srand(time(NULL));
 2       int i;
 3       for(i = 0; i < 10000; i++)
 4       {
 5           boxRGBA(screen,
 6               rand() % screen->w, rand() % screen-> h,
 7               rand() % screen->w, rand() % screen-> h,
 8               rand() % 256, rand() % 256, rand() % 256, 255);
 9           SDL_Flip(screen);
10       }
```

Najprv rozmýšľajte, potom skúste.

Keď sa pozriete do manuálu, zistíte, že podobne ako mala v predošlej lekcii funkcia `lineRGBA` svoj proťajšok `lineColor`, má aj funkcia `boxRGBA` svoj proťajšok `boxColor`. Rozdiel je opäť iba v tom, že namiesto štyroch parametrov, ktoré určujú farbu, je tam len jeden.

Ďalšie funkcie na kreslenie grafických prvkov fungujú prakticky rovnako, ako `box`. Ku každej existujú dve verzie, jedna, ktorej meno končí na `RGBA`, jedna ktorej meno končí na `Color`, ktoré sa líšia iba tým, ako sa určuje farba. V ďalšom texte budeme popisovať len ten variant s `RGBA`, ak sa vám viac hodí ten druhý, pokojne ho použite.

Funkcia `rectangleRGBA(plocha,x1,y1,x2,y2,r,g,b,alfa)` tiež kreslí na plochu, na ktorú ukazuje smerník `plocha` obdĺžniky, ktoré sú určené bodmi `[x1,y1]` a `[x2,y2]` a majú farbu určenú hodnotami `r`, `g`, `b` a `alfa`. Rozdiel oproti funkcii `box` je len v tom, že obdĺžniky nebudú vyplnené.

Na kreslenie trojuholníkov sú k dispozícii tri funkcie. Funkcia `filledTrigonRGBA(plocha,x1,y1,x2,y2,x3,y3,r,g,b,alfa)` nakreslí na plochu určenú smerníkom `plocha` vyplnený trojukolník s vrcholmi `[x1,y1]`, `[x2,y2]` a `[x3,y3]` farbou `r`, `g`, `b`, `alfa`. Funkcie `trigonRGBA` a `aatrigonRGBA` majú rovnaké parametre ako `filledTrigonRGBA` a kreslia nevyplnený trojuholník. Tá druhá z nich používa vyhladené čiary.

Funkcia `circleRGBA(plocha,x,y,polomer,r,g,b,alfa)` bude kresliť na plochu danú smerníkom `plocha` nevyplnený kruh, ktorý bude mať stred `[x,y]`, polomer `polomer` a farbu `r`, `g`, `b`, `alfa`. Funkcia `filledCircleRGBA` bude kresliť vyplnený kruh a funkcia `aacircleRGBA` vyhladený prázdny kruh.

13 Všímavejší čitatelia si iste uvedomili, že keď tie súradnice vyrábame náhodne, tak nemáme nijak zaručené, že ten prvý bod bude vľavo hore a ten druhý vpravo dole. Našťastie je funkcia `boxRGBA` rozumná. Stačí jej, že vie, že tie dva body majú určovať uhlopriečku obdĺžnika a ona už k tomu správny obdĺžnik vymyslí.

Ďalšou zaujímavou funkciou je funkcia `polygon`, ktorá sa opäť vyskytuje vo všetkých možných verziách s „`filled`“ aj „`aa`“, „`RGBA`“ aj „`Color`“ a ktorá nám dokáže nakresliť n-uholníky. Jej použitie sa ale trochu líši od predošlých funkcií, preto si dovolíme malú ukážku:

```
1     Sint16 x[] = { 320, 440, 120, 520, 200};
2     Sint16 y[] = {  20, 460, 140, 140, 460};
3     filledPolygonRGBA(screen, x, y, 5, 255, 0, 0, 255);
4     SDL_Flip(screen);
```

Keďže pri n-uholníkoch nie je zrejmé, koľko budú mať vrcholov, nemôžeme všetky vrcholy určiť ako parametre. Namiesto toho vrcholy n-uholníka napcháme do poľa, presnejšie x-ové súradnice do jedného poľa a y-ové do druhého poľa. Na prvých dvoch riadkoch sme tie polia deklarovali aj naplnili.

Možno ste si všimli neštandardný typ `Sint16`. Sú to obyčajné celé čísla, akurát majú veľkosť iba dva bajty. Dva bajty stačia, pretože sa predpokladá, že plocha, do ktorej zapisujete, nebude mať žiaden rozmer väčší, ako 32 767. Premenné tohto typu vyžadujú všetky grafické funkcie, doteraz sme vás tým ale nezaťažovali, pretože jazyk C vie skonvertovať normálny `int` na `Sint16` automaticky. Keď sa ale použije pole, je situácia trochu komplikovanejšia. Tam, kde sa v poli dá uskladniť jeden `int`, sa vmestia až dve premenné typu `Sint16`, pretože `int` zaberá štyri bajty a `Sint16` iba dva. A keby funkcia tam, kde čaká pole premenných typu `Sint16` dostala pole `int`-ov, spôsobilo by to chaos. Preto sme museli polia deklarovať poriadne a v zhode s tým, čo funkcia očakáva.[14]

Samotná funkcia `filledPolygonRGBA` má ako prvý parameter smerník na plochu, do ktorej budeme kresliť, potom nasleduje pole s x-ovými súradnicami a pole s y-ovými súradnicami. Keďže funkcia nemá ako zistiť, aké sú polia dlhé alebo akú časť z nich hodláme použiť, povieme jej to ďalším parametrom. Ďalšie štyri premenné (alebo v prípade použitia variantu s `Color` ďalšia premenná) určujú farbu, ktorou sa má n-uholník nakresliť.

Úloha 3: S pomocou generátora náhodných čísel nakreslite 10 000 náhodných trojuholníkov náhodnej farby.

Úloha 4: Vyskúšajte si použitie funkcie `polygon` uvedené vyššie. Upravte program tak, aby tam nebola tá diera v prostriedku. Upravte program tak, aby hviezda nemala zubaté okraje, ale aby boli čiary vyhladené.

Úloha 5: Nakreslite šachovnicu aj s bielymi a čiernymi poľami.

Úloha 6: Pozrite si manuál a vyskúšajte použiť niektorú z funkcií `SDL_gfx`, o ktorých v posledných dvoch lekciách nebola zmienka.

14 Našťastie sa nemôže stať, že by sme si nevšimli, keby sme omylom deklarovali pole nesprávneho typu. Na takéto veci dáva pozor kompilátor a kompilácia by skončila chybovou hláškou. Keď budete program kompilovať, môžete si vyskúšať, akú presne chybovú hlášku kompilátor vypíše, keď zadáte nesprávny typ poľa.

4. lekcia

Obrázky
alebo „Monu Lisu vpravo hore, prosím“

Zlé jazyky tvrdia, že programovanie počítačovej grafiky je len kopírovanie obrázkov z jedného miesta na druhé. Počítačoví grafici síce vedia, že to nie je celá pravda, ale sú si dobre vedomí toho, že kus pravdy na tom je. Spomeňme napríklad prvú lekciu tohto kurzu, v ktorej sme používali funkciu `SDL_BlitSurface`, ktorá robila presne to, že obrázok uložený v jednej ploche kopírovala na druhú plochu (na obrazovku). V tejto lekcii sa budeme prácou s obrázkami venovať trochu podrobnejšie.

Obrázok 6: Prasa

Predstavte si, že máte na disku súbor `prasa.png`, v ktorom máte uložený obrázok prasaťa (môžete sa pokochať na obrázku 6). Tento obrázok by ste chceli načítať a vo vašom programe nejako použiť. Prvý drobný problém je, že samotná knižnica SDL vie pracovať iba s obrázkami vo formáte `.bmp`, čo je starý formát zo systému MS Windows, ktorý má síce tú výhodu, že je jednoducho čitateľný, ale zato zaberá na disku veľmi veľa miesta. Ak chceme načítať obrázok v nejakom rozumnejšom formáte, napríklad `.png`, `.jpg` alebo `.gif`, treba siahnuť po ďalšej podpornej knižnici.

Podporná knižnica, ktorú potrebujete je `SDL_image`. Ak ju chcete použiť, treba do zdrojového kódu medzi hlavičkové súbory pridať `#include <SDL/SDL_image.h>` a pri linkovaní treba použiť parameter `-lSDL_image`.

Knižnica `SDL_image` obsahuje úžasnú a geniálnu funkciu `IMG_Load`, ktorej dáte ako parameter meno súboru s obrázkom, ona podľa koncovky určí typ obrázka, načíta obrázok do plochy a ako výsledok vráti smerník na tú novovytvorenú plochu.

Zdalo by sa, že táto funkcia rieši všetky naše problémy. To ale nemusí byť pravda. Môže sa napríklad stať, že načítanie obrázku môže zlyhať (napríklad preto, že ten obrázok ste neuložili do správneho adresára, alebo ste dali funkcii načítať niečo vo formáte `.doc` a ona je z toho jeleň). Keďže obrázkov budeme ešte v budúcnosti načítavať veľa, spravíme si na to samostatnú funkciu a tú budeme v budúcnosti používať. Tá funkcia bude vyzerať takto:

```
 1   SDL_Surface* nacitajObrazok(char* meno)
 2   {
 3     SDL_Surface* obrazok = IMG_Load(meno);
 4     if (obrazok == NULL)
 5     {
 6       fprintf(stderr, "Obrazok %s sa nepodarilo nacitat", meno);
 7       return NULL;
 8     }
 9     SDL_Surface* vylepsenyObrazok = SDL_DisplayFormat(obrazok);
10     SDL_FreeSurface(obrazok);
11     return vylepsenyObrazok;
12   }
```

Naša funkcia sa volá `nacitajObrazok`. Ako vstup dostane meno súboru a ako výsledok má vrátiť smerník na plochu s načítaným obrázkom. Na riadku 3 v nej zavoláme spomínanú funkciu `IMG_Load`. Ak táto funkcia vráti NULL, znamená to, že sa niečo pokazilo a obrázok sa nepodarilo načítať. V tom prípade vypíšeme chybovú hlášku (nezabudnite kvôli funkcii `fprintf` pridať niekam na začiatok `#include <stdio.h>`)[15] a vrátime `NULL`, aby aj hlavný program mal ako zistiť, že sa niečo pokazilo.

Ak sa všetko načítalo správne, mohli by sme vrátiť smerník `obrazok` ako hodnotu funkcie a všetko by fungovalo tak, ako má. Situácia sa ale dá ešte zlepšiť. Funkcia `SDL_DisplayFormat` vie zadanú plochu skonvertovať do rovnakého formátu, ako má displej. Použitie takejto skonvertovanej plochy je približne trikrát rýchlejšie, než použitie pôvodnej. Takže si vyrobíme novú lepšiu plochu, pamäť, v ktorej sme mali uloženú starú na riadku 10 uvoľníme a smerník na novú plochu vrátime ako výsledok funkcie.

Z hlavného programu opäť uvedieme iba tú časť, ktorá má na starosti samotné vykresľovanie:

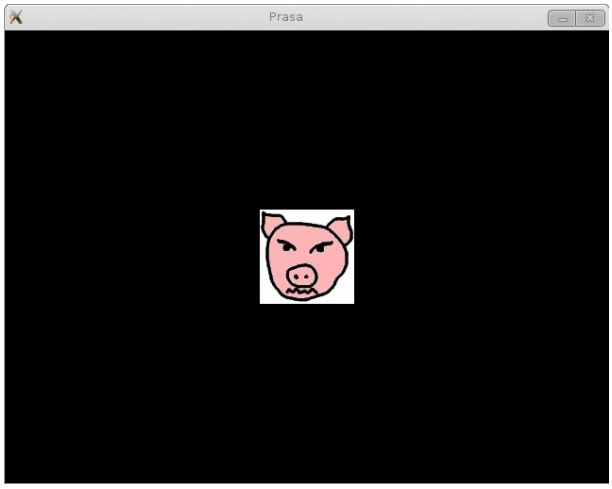
```
1     SDL_FillRect(screen,NULL,SDL_MapRGB(screen->format, 0, 0, 0));
2     SDL_Surface* prasa = nacitajObrazok("prasa.png");
3     SDL_Rect offset;
4     offset.x = (screen->w - prasa->w) / 2;
5     offset.y = (screen->h - prasa->h) / 2;
6     SDL_BlitSurface(prasa,NULL,screen,&offset);
7     SDL_Flip(screen);
```

Na prvom riadku plochu zmažeme. Na druhom nám funkcia, ktorú sme si práve napísali načíta do plochy obrázok. Zvyšok je rovnaký, ako keď sme v prvej lekcii vykresľovali nadpis. Vypočítali sme si polohu, kam chceme prasa na obrazovku skopírovať a pomocou funkcie `SDL_BlitSurface` sme ho tam aj skopírovali.

Úloha 1: Vyskúšajte. Upravte hlavný program tak, aby sa nepokúšal o kreslenie ani nečakal, ale aby rovno skončil, keď sa nepodarí obrázok načítať.

Úloha 2: Vydláždite prasatami celé okno. Použite dva cykly, jeden vnorený do druhého.

15 Na riadku 6 sme chybu vypisovali do súboru `stderr`. To je podobná vec ako `stdout`, čo je súbor, do ktorého sa vypisuje štandardný výstup. Veci, čo do `stderr` napíšete sa tiež zobrazia na termináli. Ale budú sa tam zobrazovať aj vtedy, keď si napríklad štandardný výstup presmerujete do nejakého súboru. Detaily si môžete pozrieť napríklad na tejto linke: http://osa.fiit.stuba.sk/os/html/2Zaklady.html



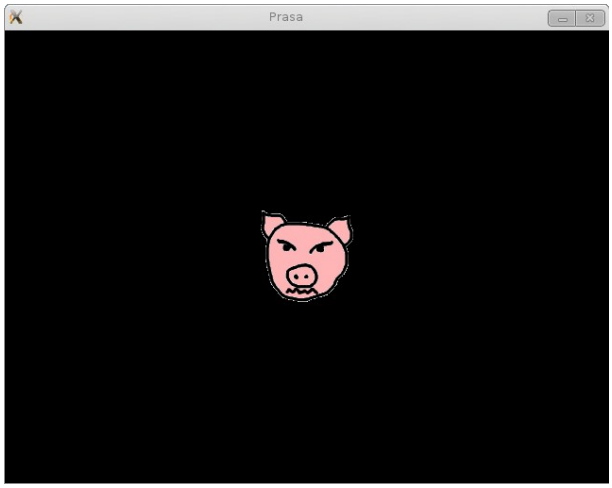
Obrázok 7: Prasa v okne

Keď programujete hru, bežne sa stáva, že potrebujete, aby sa po nejakom pozadí pohybovala postava. Pozadie máte v jednej ploche, figúrku v druhej a vždy kopírujete pozadie aj figúrku niekam na obrazovku. Problém ale je, že postavy v hrách väčšinou nebývajú obdĺžnikové. A keď jednu plochu kopírujeme na druhú, skopíruje sa celá vrátane pozadia tak, ako to môžete vidieť na obrázku 7. Potrebujeme skrátka zabezpečiť, aby sa niektoré časti plochy s postavou kopírovali a niektoré zostali priesvitné. Môžeme použiť jednu z dvoch možností:

Môžeme použiť farebný kľúč (po anglicky color key). To znamená, že obrázku povieme, ktorá jeho farba bude pokladaná za priesvitnú. V prípade nášho prasaťa môžeme napríklad programu povedať: „všetko, čo je biele, pokladaj za priesvitné“. Spraví sa to takto:

```
1     SDL_Surface* prasa = nacitajObrazok("prasa.png");
2     Uint32 colorkey = SDL_MapRGB( prasa->format, 0xFF, 0xFF, 0xFF );
3     SDL_SetColorKey( prasa, SDL_SRCCOLORKEY, colorkey );
```

Hneď potom, ako sme načítali obrázok sme si vytvorili premennú `colorkey`, do ktorej sme uložili farbu. Funkciu SDL_MapRGB sme už stretli v prvej lekcii, tiež sme potrebovali vyrobiť bielu, jediný rozdiel je v tom, že teraz machrujeme a namiesto `255` tam všade píšeme `0xFF`, aby bolo vidno, že vieme aj šestnástkovú sústavu. Na treťom riadku nastavíme ploche `prasa` farebný kľúč funkciou `SDL_SetColorKey`. Prvý parameter je smerník na plochu, ktorej ideme kľúč nastaviť, druhý je `SDL_SRCCOLORKEY`, ak chceme kľúč zapnúť alebo `0`, ak chceme kľúč vypnúť a tretí je farba, ktorú budeme pokladať za priesvitnú. Keď teraz necháte plochu vykresliť, bude to vyzerať tak, ako na obrázku 8.

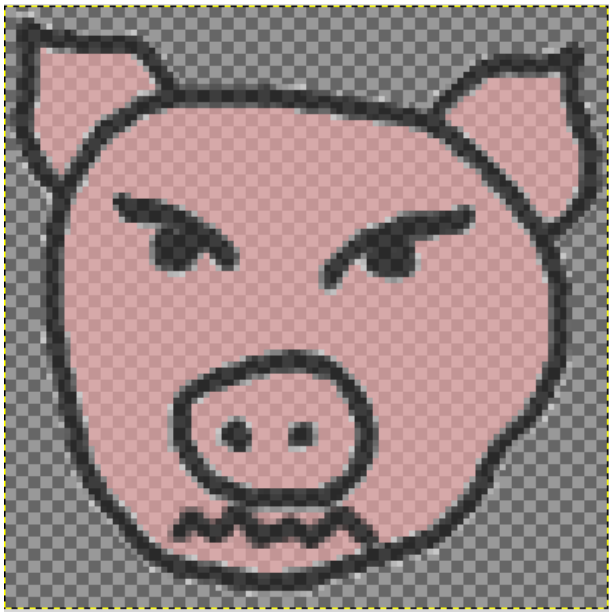


Obrázok 8: Prasa v okne bez okolia

Možno ste si všimli bielych bodiek okolo prasaťa na predošlom obrázku. Tie sú spôsobené tým, že za priesvitnú sa považuje skutočne iba farba s farebnými zložkami 255,255,255. Keby ste na obrázku mali napríklad farbu so zložkami 255,255,254, voľným okom ju od bielej nerozoznáte, ale priesvitná nebude. To sa dá niekedy využiť v náš prospech. Keď ale biele bodky nechceme, je treba mať pri kreslení prasaťa v grafickom programe vypnutý antialiasing, aby sa čierny obrys s bielym pozadím nemiešal a neovplyvňoval jeho belosť.

Vo všeobecnosti je lepšie používať ako kľúčovú takú farbu, ktorá sa v obrázkoch príliš často nevyskytuje, napr. žiarivo fialovú (255,0,255) alebo tyrkysovú (0,255,255).

Druhá možnosť je, že použijeme obrázky, ktoré už v sebe informáciu o priesvitnosti obsahujú. Odporúčame použiť napr. formát `.png`, ktorý to dokáže. Tento prístup má tú výhodu, že nemáte iba dva možné stavy: je priesvitný – nie je priesvitný, ale priesvitnosť môže mať rovnako ako každá farebná zložka až 256 úrovní.



Obrázok 9: Priesvitné prasa

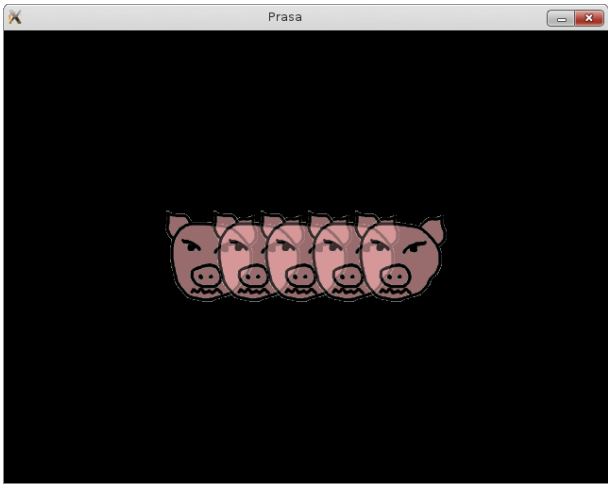
Prasa upravované v programe GIMP, ktoré môžete vidieť na obrázku 9 má hodnotu alfa nastavenú na 60 %, teda asi 154. (60 % z 255 je asi 154.) Okolie prasaťa je úplne priesvitné, teda má alfu nula.

Takéto obrázky sa používajú úplne rovnako, ako ktorékoľvek iné, len treba dať pozor na jeden detail. Keď vyrábate z načítanej plochy vylepšenú, nesmiete použiť funkciu `SDL_DisplayFormat()`, ale `SDL_DisplayFormatAlpha()`, pretože prvá informácie o priesvitnosti zahodí a druhá ich zachová. V závislosti od toho, ktorý spôsob budete používať, si upravte funkciu `nacitajObrazok`.

Výhoda tohto druhého spôsobu je, že máme viacero stupňov priesvitnosti a netreba sa babrať s kľúčmi. Výhoda prvého spôsobu je, že je menej pamäťovo náročný a je trochu rýchlejší.

Úloha 3: Vyskúšajte si obe metódy.

Úloha 4: Nakreslite na obrazovku štyridsať náhodne umiestnených polopriesvitných prasiat.



Obrázok 10: Priesvitné prasatá

5. lekcia

Animácia
alebo „Ono sa to hýbe“

Naučili sme sa, ako s pomocou knižnice SDL dostať na obrazovku to, čo potrebujete. Na to, aby z toho bola počítačová hra, je to ale stále dosť málo.[16] V hre by sa malo niečo pohybovať. Či už je to útvar zo štyroch štvorčekov, ktorý padá na spodok riadku, útočiaci zákerný nepriateľ, strela likvidujúca zákerného nepriateľa alebo útvar zo štyroch štvorčekov padajúci na zákerného nepriateľa. A pohyb vyžaduje animáciu.

Princíp animácie je jednoduchý. To, čo potrebujeme, nakreslíme niekam na obrazovku, o chvíľu to nakreslíme niekam inam a o ďalšiu chvíľu niekam ešte ďalej. Keď to budeme robiť šikovne, bude efekt rovnaký, ako keď sa pozeráte na film. Tam vám tiež ukážu za sekundu dvadsaťpäť obrázkov a máte pocit, že sa veci hýbu.

Prvá ukážka bude jednoduchšia. Necháme po obrazovke pobehovať koliesko. Keď koliesko príde na kraj okna, odrazí sa od neho a bude pokračovať na druhú stranu. Nech sa páči, zdrojový kód:

```
 1  #include <SDL/SDL.h>
 2  #include <SDL/SDL_gfxPrimitives.h>
 3  #include <time.h>
 4
 5  typedef struct koliesko {
 6    int x, y;  // poloha
 7    int vx, vy; // rychlost
 8    int rad;  // polomer
 9    int r, g, b, a; // farba
10  } KOLIESKO;
11
12  void inicializujKoliesko(KOLIESKO *k, SDL_Surface *screen)
13  {
14    k->x = rand() % screen->h;
15    k->y = rand() % screen->w;
16    k->vx = rand() % 10;
17    k->vy = rand() % 10;
18    k->rad = 20 + rand() % 10;
19    k->r = rand() % 256;
20    k->g = rand() % 256;
21    k->b = rand() % 256;
22    k->a = 255;
23  }
24
25  void kresliKoliesko(KOLIESKO *k, SDL_Surface *screen)
26  {
27    filledCircleRGBA(screen,k->x,k->y,k->rad,k->r,k->g,k->b,k->a);
28  }
29
30  void pohniKoliesko(KOLIESKO *k, SDL_Surface *screen)
```

16 Ono aj s takýmto málom sa dajú spraviť zaujímavé veci. Je celá kategória hier nazvaná „visual novels“ populárna najmä v Japonsku, ktorej oveľa viac netreba.

```
31  {
32    k->x = k->x + k->vx;
33    k->y = k->y + k->vy;
34    if (k->x > screen->w || k->x < 0)
35      k->vx = -(k->vx);
36    if (k->y > screen->h || k->y < 0)
37      k->vy = -(k->vy);
38  }
39
40  int main(int argc, char *argv[])
41  {
42    SDL_Surface *screen = NULL;
43    KOLIESKO kol;
44
45    SDL_Init( SDL_INIT_EVERYTHING );
46
47    screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE);
48    SDL_WM_SetCaption( "Koliesko", NULL );
49    srand(time(NULL));
50
51    inicializujKoliesko(&kol,screen);
52
53    while (1)
54    {
55      SDL_Event event;
56      if (SDL_PollEvent( &event ))
57        if ((event.type == SDL_QUIT) ||
58              (event.type == SDL_KEYDOWN &&
59               event.key.keysym.sym == SDLK_ESCAPE))
60            break;
61      SDL_Delay(5);
62      SDL_FillRect(screen,NULL,SDL_MapRGB(screen->format, 0, 0, 0));
63      pohniKoliesko(&kol, screen);
64      kresliKoliesko(&kol, screen);
65      SDL_Flip(screen);
66    }
67    SDL_Quit();
68    return 0;
69  }
```

Prvá zaujímavá vec sa udeje na riadkoch 5 až 10. Vyrobíme si tam štruktúru `KOLIESKO`, v ktorej budeme mať uložené všetky informácie o koliesku – jeho súradnice, rýchlosť, polomer a farbu. Takýto prístup má viacero výhod. V prvom rade máme všetky údaje pekne pohromade a nie rozťahané po deviatich rôznych premenných. To nám umožňuje vyrobiť si funkcie, ktoré ako parameter dostanú namiesto dlhého zoznamu premenných jeden smerník na premennú typu `KOLIESKO` a môžu s tým kolieskom spraviť všetko, čo potrebujú. Druhá veľká výhoda je to, že keby sme chceli mať na obrazovke dve kolieska naraz, nemusíme vyrábať ďalších deväť premenných, ale stačí vytvoriť jedno nové koliesko.

Na riadkoch 12 až 23 môžete vidieť funkciu `inicializujKoliesko`, ktorá má za úlohu spraviť koliesku všetky úvodné nastavenia. Táto funkcia má dva vstupné parametre. Prvý je smerník na koliesko, ktorému má nastaviť vstupné hodnoty, druhý je smerník na plochu okna. Tú plochu okna tam potrebujeme na to, aby sme nastavili pozíciu kolieska niekde vo vnútri okna a nie mimo

(pozrite si riadky 14 a 15). Ostatné premenné nastavujeme náhodne. Polomer kolieska bude mať náhodnú hodnotu od 20 do 29 (Užitočná finta na riadku 18!) a priesvitnosť bude vždy 255.

Na riadkoch 25 až 28 nájdete funkciu `kresliKoliesko`. Opäť má dva parametre – smerník na koliesko a smerník na plochu a jej jediným účelom je koliesko nakresliť. Môže sa zdať, že je zbytočné robiť funkciu, ktorá obsahuje jediný príkaz. Odmenou vám ale bude výrazne čitateľnejšia hlavná funkcia. Okrem toho v budúcnosti neostanete len pri kolieskach. Kresliace funkcie môžu byť oveľa zložitejšie. A je dobré držať si ich na samostatnom mieste.

Na riadkoch 30 až 38 je funkcia `pohniKoliesko`, ktorá má za úlohu vypočítať nové súradnice kolieska. Robí to jednoducho. K súradnici `x` pripočíta rýchlosť v horizontálnom smere `vx` a k súradnici `y` pripočíta rýchlosť vo vertikálnom smere `vy`. Problém nastane iba vtedy, keby nám koliesko chcelo ujsť z okna. Na riadku 34 sa pozrieme, či nám koliesko náhodou nepreliezlo pravý alebo ľavý okraj a ak náhodou áno, otočíme rýchlosť v horizontálnom smere na opačnú. Podobne na riadku 36 skontrolujeme, či koliesko nechce ujsť hore alebo dole a ak áno, tak mu otočíme vertikálnu rýchlosť.

Keď máme toto všetko hotové, môžeme písať hlavnú funkciu. Na riadku 42 si deklarujeme plochu okna a na riadku 43 premennú `kol` typu `KOLIESKO`. Na riadkoch 45 až 49 štartujeme štandardné veci a nastavujeme generátor náhodných čísel. Na riadku 51 nastavíme koliesku všetky počiatočné hodnoty.

Celá animácia sa odohráva v cykle na riadkoch 53 až 66. Je to cyklus zjavne nekonečný, pretože podmienka je vždy pravda. Na začiatku sa pozrieme, či niekto nezavrel okno, alebo nestlačil klávesu ESC a v prípade, že sa tak stalo, príkazom `break` nekonečný cyklus ukončíme. Ak si ale používateľ nášho programu nepraje skončiť, udejú sa riadky 62 až 65. Tam vyfarbíme plochu na čierno, vypočítame novú polohu kolieska, nakreslíme ho a novú plochu zobrazíme v okne. A toto opakujeme, kým to používateľa neprestane baviť.

Keď nekonečný cyklus skončí, upraceme s pomocou funkcie `SDL_Quit` a skončíme celý program.

Úloha 1: Pochopte a vyskúšajte. Pri kompilácii nezabudnite prilinkovať `SDL_gfx`.

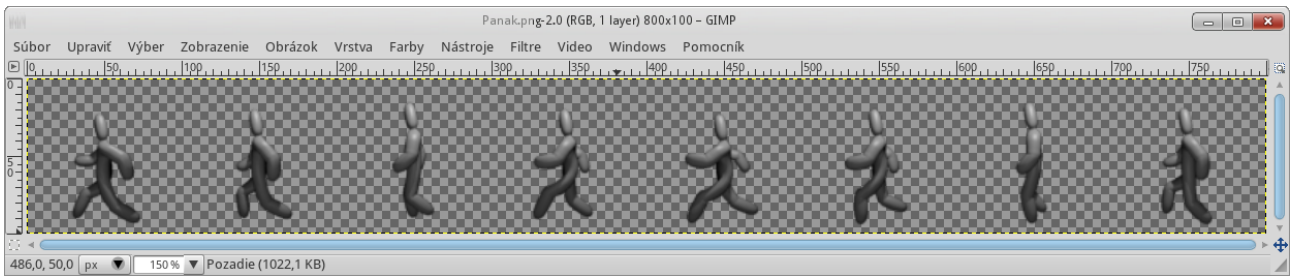
Úloha 2: Vo funkcii `inicializujKoliesko` nastavujeme rýchlosť v horizontálnom aj vertikálnom smere na náhodné hodnoty od 0 do 9. To znamená, že obe hodnoty budú nezáporné a teda na začiatku vyrazí koliesko niekde smerom vpravo dole. Upravte procedúru tak, aby bola rýchlosť v každom smere náhodná hodnota od -10 do 10.

Úloha 3: Upravte program tak, aby po ploche pobehovali štyri kolieska.

Keď sa po ploche pohybuje koliesko, situácia je celkom jednoduchá. Stačí ho kresliť na rôzne miesta a vec je vybavená. Trochu komplikovanejšia situácia nastane, keď chceme, aby nám po ploche pobehovala nejaká postava.

Vtedy bude treba najprv spraviť kúsok grafickej roboty a urobiť si animáciu postavy – sériu rovnako veľkých obrázkov, ktoré znázorňujú jednotlivé fázy pohybu postavy. Môžete použiť grafický nástroj vášmu srdcu milý, napríklad GIMP, my sme použili Blender. Môžete rovno vyrobiť obrázky s priesvitným pozadím alebo môžete neskôr použiť farebný kľúč spôsobom, ktorý bol opísaný v predošlej lekcii.

Keď obrázky máme, treba si ich všetky vložiť do jedného dlhého obrázka. Ten náš môžete vidieť na obrázku 11. Každý z našich obrázkov mal rozmer 100×100 pixelov, máme osem fáz, takže výsledný obrázok má veľkosť 800×100 pixelov. Názov súboru bude `Panak.png`.



Obrázok 11: Animácia figúrky

Teraz už len stačí napísať program, ktorý bude našu skvelú animáciu používať. Zdrojový kód sa bude štruktúrou podobať na predošlý program a môže vyzerať takto:

```
1   #include <SDL/SDL.h>
2   #include <SDL/SDL_image.h>
3
4   typedef struct panak {
5     SDL_Surface* obrazok;
6     int sirkaFazy;
7     int faza;
8     int pocetFaz;
9     int x,y;    // poloha
10    int vx,vy;
11  } PANAK;
12
13  SDL_Surface* nacitajObrazok(char* meno)
14  {
15    SDL_Surface* obrazok = IMG_Load(meno);
16    if (obrazok == NULL)
17    {
18      fprintf(stderr, "Obrazok %s sa nepodarilo nacitat", meno);
19      return NULL;
20    }
21    SDL_Surface* vylepsenyObrazok = SDL_DisplayFormatAlpha(obrazok);
22    SDL_FreeSurface(obrazok);
23    return vylepsenyObrazok;
24  }
25
26  void inicializujPanaka(PANAK* pp, SDL_Surface* screen)
27  {
28    pp->obrazok = nacitajObrazok("Panak.png");
29    pp->sirkaFazy = 100;
30    pp->faza = 0;
31    pp->pocetFaz = 8;
32    pp->x = screen->w;
33    pp->y = screen->h - pp->obrazok->h - 10;
34    pp->vx = -10;
35    pp->vy = 0;
36  }
37
38
39
40
```

```
41  void kresliPanaka(PANAK* pp, SDL_Surface* screen)
42  {
43    SDL_Rect odkial, kam;
44    odkial.x = pp->faza * pp->sirkaFazy;
45    odkial.y = 0;
46    odkial.h = pp->obrazok->h;
47    odkial.w = pp->sirkaFazy;
48    kam.x = pp->x;
49    kam.y = pp->y;
50    SDL_BlitSurface(pp->obrazok,&odkial,screen,&kam);
51  }
52
53  void pohniPanaka(PANAK* pp, SDL_Surface* screen)
54  {
55    pp->x += pp->vx;
56    pp->y += pp->vy;
57    if (pp->x < -100)
58      pp->x = screen->w;
59    pp->faza = (pp->faza + 1 ) % pp->pocetFaz;
60  }
61
62  int main(int argc, char *argv[])
63  {
64    SDL_Surface *screen = NULL;
65    PANAK jozin;
66
67    SDL_Init( SDL_INIT_EVERYTHING );
68
69    screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE);
70    SDL_WM_SetCaption( "Panak", NULL );
71
72    inicializujPanaka(&jozin, screen);
73
74    while (1)
75    {
76      SDL_Event event;
77      if (SDL_PollEvent( &event ))
78        if ((event.type == SDL_QUIT) ||
79              (event.type == SDL_KEYDOWN &&
80               event.key.keysym.sym == SDLK_ESCAPE))
81            break;
82      SDL_Delay(100);
83      SDL_FillRect(screen,NULL,SDL_MapRGB(screen->format, 0, 0, 0));
84      pohniPanaka(&jozin, screen);
85      kresliPanaka(&jozin, screen);
86      SDL_Flip(screen);
87    }
88    SDL_FreeSurface(jozin.obrazok);
89    SDL_Quit();
90    return 0;
91  }
```

Tak ako predtým aj teraz si urobíme štruktúru, ktorá bude opisovať stav panáka. Nazýva sa `PANAK` a vidíte ju na riadkoch 4 až 11. Podobne ako pri koliesku si budeme pamätať aktuálnu

polohu panáka (položky `x` a `y`) a jeho rýchlosť (položky `vx` a `vy`). Okrem toho si budeme pamätať smerník na plochu s obrázkom `obrazok`, šírku jednej fázy `sirkaFazy` (výšku si pamätať nemusíme, lebo je rovnaká, ako výška celého obrázka), aktuálnu fázu, v ktorej sa panák práve nachádza `faza` a počet fáz animácie `pocetFaz`.

Funkcia `nacitajObrazok` je úplne rovnaká ako v predošlej lekcii a budeme ju používať na načítanie obrázku do správneho typu plochy. Z priestorových dôvodov neošetrujeme situáciu, keď sa obrázok nepodarí načítať a funkcia vráti `NULL`.

Funkcia `inicializujPanaka` (riadky 26 až 36) nastavuje panáka. To, čo sa v nej deje je pomerne zrejmé a netreba to bližšie opisovať. Jediný zaujímavý riadok je riadok 33, kde nastavujeme polohu panáka tak, aby bol 10 pixelov nad spodkom okna. Ak ste nevedeli prečítať to `pp->obrazok->h`, znamená to „výška obrázka panáka na ktorého ukazuje smerník `pp`“.

Funkcia `kresliPanaka` (riadky 41 až 51) je zaujímavejšia. Potrebujeme si v nej vyrobiť dva obdĺžniky (`SDL_Rect`) s názvami `odkial` a `kam`, ktoré nám určia, ktorá časť obrázka sa má kopírovať do okna a kam sa má kopírovať. Zdrojový obdĺžnik nastavíme podľa toho, ktorú fázu práve chceme zobrazovať. To spravíme tak, že v riadku 44 nastavíme podľa aktuálnej fázy (a šírky fázy) hodnotu `x` obdĺžnika `odkial`. Ak bude fáza 0, `x` bude 0, ak bude fáza 1, `x` bude 100, ak bude fáza 2, `x` bude 200 atď. Hodnoty `y`, `h` a `w` obdĺžnika `odkial` budú vždy rovnaké – `y` bude vždy 0 a `w` a `h` sú rozmery jednej fázy, čiže obe budú 100. Cieľový obdĺžnik nastavíme podľa hodnôt `x` a `y` panáka. Na záver správnu fázu panáka vykreslíme funkciou `SDL_BlitSurface`.

Funkcia `pohniPanaka` zmení panákove súradnice `x` a `y` o rýchlosť. (Keďže rýchlosť panáka vo vertikálnom smere je 0, súradnicu `y` by sme nemuseli meniť. Uvádzame ale všeobecnejšiu verziu, keby ste chceli použiť funkciu v nejakom svojom programe.) Ak panák zmizne za ľavým okrajom okna, bude opäť teleportovaný vpravo (riadky 57 až 58). Nakoniec ešte treba zvýšiť fázu o 1. Keďže sú fázy číslované od nuly, číslo fázy by nemalo byť väčšie, než `pocetFaz - 1`. Preto výsledok ešte upravíme tak, aby sa v prípade, že už dosiahol hodnotu `pocetFaz` z neho opäť stala nula.

Hlavný program je až na detaily úplne rovnaký, ako v predošlom prípade.

Úloha 4: Pochopte a vyskúšajte. Pri kompilácii nezabudnite prilinkovať `SDL_image`.

Úloha 5: Zmeňte rýchlosť panáka na –3. Budete vidieť, že chôdza zrazu nie je prirodzená a vyzerá skôr ako moon walk v podaní Michaela Jacksona. Rýchlosť je v prípade chôdze zviazaná so samotnou animáciou. Ak chcete postavu zrýchliť či spomaliť, meňte radšej parameter funkcie `SDL_Delay` na riadku 82.

Úloha 6: Zmeňte program tak, aby panák, keď prejde sprava doľava vycúval a vrátil sa pospiatky zľava doprava. Nezabudnite zobrazovať fázy v opačnom poradí. Funkcii `pohniPanaka` môžete pridať ďalší parameter, podľa ktorého sa rozhodne, či pohne panákom dopredu alebo dozadu, alebo sa môžete rozhodnúť podľa toho, či má rýchlosť v smere `x` kladnú, alebo zápornú.

Úloha 7: Zmeňte program tak, aby po ploche behali naraz traja panáci, každý z nich bol inde a aby nemali rovnakú fázu. Pokúste sa to urobiť tak, aby ste plochu s obrázkom načítali len raz a aby všetci panáci používali tú istú plochu. Funkcii `inicializujPanaka` môžete pridať ďalší parameter.

Úloha 8: (Nepovinná, pre guruov.) Ošetrite načítanie obrázka tak, aby program skončil korektne, keď sa obrázok nepodarí načítať.

6. lekcia
Klávesnica
alebo „Udalosť roka“

Keď ste vzorne naštudovali všetky doterajšie lekcie, viete dostať na obrazovku prakticky čokoľvek a dokonca viete zariadiť, aby sa to hýbalo. Na to, aby z vášho programu vznikla napríklad nejaká hra je to ale ešte stále málo. Problém je v tom, že používateľ zatiaľ nemá veľa možností do činnosti vášho programu zasiahnuť, takže sa to podobá skôr na kino, než na hru.

Na to, aby ste mohli nejako zmysluplne s programom komunikovať, musí program vnímať, čo s ním chcete robiť. Takéto vnímanie sa väčšinou deje prostredníctvom udalostí. Udalosti, ktoré sa udejú, sú ukladané do fronty. Ak chcete v programe zistiť, čo používateľ vlastne chce, treba sa do tej fronty pozrieť a udalosti spracovať. To sme samozrejme robili aj v našich doterajších programoch – inak by nevedeli, že majú skončiť. V tejto lekcii si ale ukážeme, ako to spraviť poriadne.

Udalostí existuje viacero druhov. V tejto lekcii sa budeme venovať udalostiam pochádzajúcim od klávesnice, pričom ukážeme aj iný spôsob, ako s klávesnicou pracovať. Tradične najprv predvedieme ukážkový program. V okne programu sa vykreslí koliesko, ktorým budeme môcť pohybovať pomocou šípok a pomocou klávesov `v` a `m` ho budeme vedieť zväčšiť alebo zmenšiť. V komentári pod programom sa dozviete o ďalších možnostiach spracovania udalostí, ktoré máte k dispozícii.

```
 1  #include <SDL/SDL.h>
 2  #include <SDL/SDL_gfxPrimitives.h>
 3  #include <time.h>
 4
 5  typedef struct koliesko {
 6    int x, y;  // poloha
 7    int rad;  // polomer
 8    int r, g, b, a; // farba
 9  } KOLIESKO;
10
11  void inicializujKoliesko(KOLIESKO *k, SDL_Surface *screen)
12  {
13    k->x = screen->w / 2;
14    k->y = screen->h / 2;
15    k->rad = 20 + rand() % 10;
16    k->r = rand() % 256;
17    k->g = rand() % 256;
18    k->b = rand() % 256;
19    k->a = 255;
20  }
21
22  void kresliKoliesko(KOLIESKO *k, SDL_Surface *screen)
23  {
24    filledCircleRGBA(screen,k->x,k->y,k->rad,k->r,k->g,k->b,k->a);
25  }
26
27
28
```

```
29  void pohniKoliesko(KOLIESKO *k, SDL_Surface *screen,
30                                                  int dx, int dy)
31  {
32    if ((k->x + dx < screen->w) && (k->x + dx >= 0))
33      k->x = k->x + dx;
34    if ((k->y + dy < screen->h) && (k->y + dy >= 0))
35      k->y = k->y + dy;
36  }
37
38  int main(int argc, char *argv[])
39  {
40    SDL_Surface *screen = NULL;
41    KOLIESKO kol;
42    int koncime = 0;
43
44    SDL_Init( SDL_INIT_EVERYTHING );
45
46    screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE);
47    SDL_WM_SetCaption( "Ovladnute koliesko", NULL );
48    srand(time(NULL));
49
50    inicializujKoliesko(&kol,screen);
51
52    while (koncime == 0)
53    {
54      SDL_Event event;
55      while (SDL_PollEvent( &event ))
56      {
57        if (event.type == SDL_QUIT)
58           koncime = 1;
59        if (event.type == SDL_KEYDOWN)
60        {
61          switch (event.key.keysym.sym)
62           {
63             case SDLK_ESCAPE:
64               koncime = 1;
65               break;
66             case SDLK_m:
67               if (kol.rad > 5)
68                 kol.rad--;
69               break;
70             case SDLK_v:
71               if (kol.rad < 200)
72                 kol.rad++;
73               break;
74          }
75        }
76      }
```

```
77        Uint8 *stavKlavesnice = SDL_GetKeyState( NULL );
78        if (stavKlavesnice[SDLK_UP])
79          pohniKoliesko(&kol,screen,0,-1);
80        if (stavKlavesnice[SDLK_DOWN])
81          pohniKoliesko(&kol,screen,0,1);
82        if (stavKlavesnice[SDLK_LEFT])
83          pohniKoliesko(&kol,screen,-1,0);
84        if (stavKlavesnice[SDLK_RIGHT])
85          pohniKoliesko(&kol,screen,1,0);
86
87
88        SDL_FillRect(screen,NULL,SDL_MapRGB(screen->format, 0, 0, 0));
89        kresliKoliesko(&kol, screen);
90        SDL_Flip(screen);
91      }
92      SDL_Quit();
93      return 0;
94    }
```

Vytvoríme si štruktúru `KOLIESKO`, ktorá si o koliesku bude pamätať všetko dôležité a napíšeme pre ňu funkcie `inicializujKoliesko`, `kresliKoliesko` a `pohniKoliesko` podobne, ako sme to spravili v predošlej lekcii. Všimnite si, že funkcia `inicializujKoliesko` nastaví koliesko do stredu okna a funkcia `pohniKoliesko` dostane na vstupe dva celočíselné parametre `x` a `y`, aby vedela, o koľko má v jednotlivých smeroch to koliesko pohnúť.

Všetko podstatné sa udeje vo funkcii `main`, konkrétne v cykle, ktorý začína na riadku 52 a končí na riadku 91. Samotný cyklus je riadený premennou `koncime`. Táto premenná bola na riadku 42 nastavená na hodnotu 0 a cyklus sa bude opakovať dovtedy, kým sa hodnota tejto premennej nezmení.

Na riadku 54 je deklarovaná premenná `event` typu `SDL_Event`. V tomto type premenných vie knižnica SDL uložiť všetky detaily o nejakej konkrétnej udalosti.

Cyklus na riadkoch 55 až 76 spracuje všetky udalosti, ktoré sú momentálne k dispozícii. Príkaz `SDL_PollEvent( &event )` spôsobí, že sa z fronty udalostí vyberie najstaršia nevyhodnotená udalosť a vloží sa do premennej `event`. V prípade, že už vo fronte žiadna ďalšia udalosť nie je (buď preto, lebo sme už všetky spracovali, alebo preto, že sa nič nové nestalo), funkcia `SDL_PollEvent` vráti hodnotu 0, takže cyklus skončí.

V samotnom cykle sa vždy pozrieme, aká udalosť nastala a podľa toho sa zariadime. V prvom rade určujeme typ udalosti. Typov môže byť mnoho.[17] V našom prípade pozeráme, či nenastal typ udalosti `SDL_QUIT`, ktorý hovorí, že niekto ukončil program (napríklad tak, že zavrel okno) alebo či nenastala udalosť `SDL_KEYDOWN`, teda či nebol stlačený nejaký kláves.

Keď už vieme, že udalosť bola typu `SDL_KEYDOWN`, môžeme sa o nej dozvedieť ďalšie podstatné detaily. Pre každý typ udalosti sa ale príslušné detaily skrývajú v inej položke jej dátovej štruktúry. Pre udalosti typu `SDL_KEYDOWN` a `SDL_KEYUP` sa nachádzajú v položke `key` (takže v našom programe k nim budeme pristupovať cez `event.key`). Ak budete potrebovať vedieť, kde hľadať informácie o ostatných typoch udalostí, pozrite sa na stránku http://sdl.beuc.net/sdl.wiki/SDL_Event. Dozviete sa tam napríklad aj to, že k popisu udalostí týkajúcich sa pohybu myši sa dostanete cez `event.motion`.

17 Úplný zoznam nájdete na http://sdl.beuc.net/sdl.wiki/SDL_Event

Ale čo je ešte dôležitejšie, nájdete tam linku na dátovú štruktúru, ktorá opisuje konkrétnu udalosť. Pre každú udalosť sa totiž pamätajú trochu iné veci. Pri kliknutí myšou napríklad potrebujete vedieť, kde a ktorým tlačidlom sa klikalo, pri pohybe myši potrebujete poznať súradnice kurzora a pri zmene veľkosti okna potrebujete vedieť jeho nové rozmery. Keď sa teda pozriete, v akej štruktúre sa popisujú dáta pre udalosť typu `SDL_KEYDOWN`, zistíte, že v štruktúre `SDL_KeyboardEvent`. A keď kliknete na príslušnú linku, zistíte, čo sa všetko o udalosti môžete dozvedieť.

Štruktúra `SDL_WindowEvent` obsahuje tri položky: `type`, `state` a `keysym`. Položka `type` hovorí, aká udalosť nastala (a môže mať hodnotu `SDL_KEYDOWN` alebo `SDL_KEYUP`). Položka `state` hovorí, v akom stave bol kláves po udalosti – či bol stlačený (`SDL_PRESSED`) alebo uvoľnený (`SDL_RELEASED`). Nás ale zaujíma tretia položka – `keysym` – ktorá nám prezradí, aký kláves bol vlastne stlačený.

Položka `keysym` je opäť štruktúra, ktorá obsahuje ďalšie štyri položky: `scancode`, `sym`, `mod` a `unicode`. Položku `scancode` pravdepodobne veľmi nevyužijete – to, aký kód pri stlačení vráti, závisí od konkrétneho hardvéru. Podobne položku `unicode` radšej nepoužívajte.[18]

Zato druhé dve položky sú použiteľné značne. Položka `mod` hovorí, či bol počas stlačenia klávesu stlačená niektorý modifikačný kláves. Ak chceme teda napríklad zistiť, či niekto počas stlačenia klávesu držal pravý Alt, spravíme to takto:

```
if (event.key.keysym.mod & KMOD_RALT) printf("Lavy alt bol drzany");
```

Modifikátorov je samozrejme viacero. Všetky modifikátory, ktoré sú k dispozícii, nájdete na stránke http://sdl.beuc.net/sdl.wiki/SDL_GetModState.

Možno ste si všimli zaujímavý spôsob, akým sme napísali podmienku v príklade. Ide o to, že tých modifikátorov môže byť stlačených viacero a nás zaujíma, či je medzi nimi aj ten pravý Alt. Keby sme podmienku napísali iba ako `(event.key.keysym.mod == KMOD_RALT)` tak by nám to tie ostatné modifikátory mohli pokaziť.[19]

A konečne sa dostávame k položke pre nás najpodstatnejšej, k položke `sym`, ktorá obsahuje kód tlačidla, ktoré práve bolo stlačené. Kódy jednotlivých tlačidiel nájdete na stránke http://sdl.beuc.net/sdl.wiki/SDLKey. Na riadkoch 61 až 74 sa pozrieme, ktorý kláves bol vlastne stlačený. Ak to bol `Escape` (hodnota `SDLK_ESCAPE`), dáme vedieť programu, že má skončiť, ak to bol kláves `m` (hodnota `SDLK_m`), koliesko zmenšíme (dáme pri tom pozor, aby polomer nebol menší, než 5 pixelov) a ak to bol kláves `v` (hodnota `SDLK_v`), koliesko zväčšíme.

---

18 A keď ju už silou-mocou používať chcete, tak si pozrite v manuáli, ako sa to robí.

19 Na to, aby ste presne rozumeli, ako to funguje, treba povedať niečo o dvojkovej sústave. SDL totiž potrebuje uložiť stavy všetkých stlačených modifikátorov do jednej premennej. A tak sa na tú premennú pozrie ako na číslo v dvojkovej sústave, každej cifre vyhradí jeden modifikátor a tie cifry, ktorých modifikátory sú stlačené nastaví na 1. Keby sme stlačili úplne všetky (všetky na vašej klávesnici pravdepodobne nebudú), tak to číslo bude vyzerať takto: `0111 1111 1100 0011`, pričom jednotky majú v uvedenom poradí nasledujúci význam: `Mode`, `Caps`, `Num`, `Pravý Meta`, `Ľavý Meta`, `Pravý Alt`, `Ľavý Alt`, `Pravý Ctrl`, `Ľavý Ctrl`, `Pravý Shift`, `Ľavý Shift`. Keď teraz chceme zistiť, či je stlačený pravý `Alt`, zoberieme konštantu `KMOD_RALT`, ktorá má v dvojkovej sústave zápis `0000 0010 0000 0000` a použijeme operáciu `&`, čo znamená „bitový and“. Operácia prejde po jednotlivých cifrách obe čísla a do výsledku dá na patričné miesto jednotku iba vtedy, ak boli na danom mieste v oboch originálnych jednotky. Keďže však `KMOD_RALT` má jednotku len na jedinom mieste, výsledok nebude nula iba vtedy, keď má jednotku na tom istom mieste aj `event.key.keysym.mod`, čo znamená, že je stlačený pravý Alt. Ak by sme namiesto `event.key.keysym.mod & KMOD_RALT` použili `event.key.keysym.mod == KMOD_RALT`, tak ak by sme okrem pravého `Alt`u stlačili ešte nejaký iný modifikátor, podmienka by sa vyhodnotila ako nepravdivá.

Tento spôsob ovládania pomocou klávesnice ale nie je vždy presne to, čo potrebujeme. Keď chceme koliesko zväčšiť o sto pixelov, musíme kláves `v` stlačiť stokrát, čo môže byť celkom otrava. Keby sme mali podobným spôsobom kolieskom pohybovať, tak by to bolo ešte nepríjemnejšie.

Ak máme kolieskom nejak zmysluplne hýbať, oveľa viac by sa nám hodilo, keby sme sa mohli pozrieť, že či je práve stlačená šípka hore. V prípade, že áno, kolieskom by sme pohli nahor, v prípade, že nie, koliesko by sme nechali tak. Vtedy by sme nemuseli kláves zbesilo stláčať, ale stačilo by ho držať a koliesko by sa pohybovalo nahor. Knižnica SDL nám našťastie umožňuje aj takýto prístup.

Funkcia, ktorú použijeme, sa nazýva `SDL_GetKeyState`. Táto funkcia vráti pole, ktorého prvky majú hodnotu 0 alebo 1 podľa toho, či je daný kláves stlačený, alebo nie. Toto pole sme si na riadku 77 uložili do premennej `stavKlavesnice`. Potom stačí pozrieť, či je hodnota `stavKlavesnice[SDLK_UP]` rôzna od nuly a vieme, že šípka hore bola stlačená. Ako indexy používame tie isté konštanty, ako pri vyhodnocovaní `event.key.keysym.sym`. Pohyb kolieska spracujeme na riadkoch 78 až 85.

A to je všetko. Na riadkoch 88 až 90 už len scénu vykreslíme.

Úloha 1: Pochopte a vyskúšajte.

Úloha 2: Upravte program tak, aby po stlačení klávesu `f` koliesko zmenilo farbu. Farbu nemá meniť celý čas, kým držíte kláves, ale iba raz.

Úloha 3: Čo bude program robiť keď stlačíte naraz šípku hore a šípku vpravo? Čo bude program robiť, keď stlačíte naraz šípku hore a šípku dole? Prečo sa tak správa?

Úloha 4: Upravte program tak, aby koliesko menilo veľkosť nie o 1, ale o 10, keď budete pri stláčaní klávesov `m` a `v` držať `Ctrl`. Dbajte na to, aby bol polomer stále minimálne 5 a maximálne 200 pixelov.

Úloha 5: Upravte program tak, aby koliesko pomaly tmavlo, keď budete držať kláves `t`.

7. lekcia

Myš alebo „Ďalšie radostné udalosti“

V predošlej lekcii sme si povedali niečo viac o udalostiach a venovali sme sa udalostiam týkajúcim sa klávesnice. Táto lekcia bude o udalostiach týkajúcich sa myši.

Práca s udalosťami od myši je v podstate rovnaká, než práca s udalosťami od klávesnice. Existujú tri typy udalostí od myši: `SDL_MOUSEBUTTONDOWN`, `SDL_MOUSEBUTTONUP` a `SDL_MOUSEMOTION`. Prvá z nich nastane, keď sa stlačí niektoré tlačidlo na myši, druhá nastane, keď sa niektoré tlačidlo myši pustí a tretia nastane, keď sa myš pohne.

Použitie týchto udalostí môžete vidieť v nasledujúcom programe. Program je určený na jednoduché kreslenie:

```
 1  #include <SDL/SDL.h>
 2  #include <SDL/SDL_gfxPrimitives.h>
 3
 4  int main(int argc, char *argv[])
 5  {
 6    SDL_Surface *screen = NULL;
 7    int koncime = 0;
 8    int oldx, oldy, newx, newy;
 9    int kreslime = 0;
10
11    SDL_Init( SDL_INIT_EVERYTHING );
12
13    screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE);
14    SDL_WM_SetCaption( "Kreslenie", NULL );
15
16    SDL_FillRect(screen,NULL,SDL_MapRGB(screen->format, 0, 0, 0));
17
18    while (koncime == 0)
19    {
20      SDL_Event event;
21      while (SDL_PollEvent( &event ))
22      {
23        switch (event.type)
24        {
25            case SDL_QUIT:
26              koncime = 1;
27              break;
28            case SDL_KEYDOWN:
29              if (event.key.keysym.sym == SDLK_ESCAPE)
30                koncime = 1;
31              break;
32            case SDL_MOUSEBUTTONDOWN:
33              if (event.button.button == SDL_BUTTON_LEFT)
34              {
35                kreslime = 1;
36                oldx = event.button.x;
37                oldy = event.button.y;
38              }
39              break;
```

```
40              case SDL_MOUSEMOTION:
41                if (kreslime)
42                {
43                  newx = event.motion.x;
44                  newy = event.motion.y;
45                  lineColor(screen,oldx,oldy,newx,newy,0xffffffff);
46                  oldx = newx;
47                  oldy = newy;
48                }
49                break;
50              case SDL_MOUSEBUTTONUP:
51                if (event.button.button == SDL_BUTTON_LEFT)
52                  kreslime = 0;
53                break;
54            }
55          }
56
57          SDL_Flip(screen);
58        }
59        SDL_Quit();
60        return 0;
61    }
```

Na riadkoch 1 až 17 sa štandardným spôsobom inicializuje knižnica SDL. Z tejto časti sú zaujímavé iba premenné, ktoré náš program bude potrebovať. Premenná `koncime` slúži ako riadiaca premenná hlavného cyklu. Cyklus bude bežať, kým je jej hodnota 0. Premenná `kreslime` bude mať hodnotu 1 vtedy, keď bude stlačené ľavé tlačidlo myši. Podľa nej budeme vedieť rozlišovať, či práve kreslíme, alebo nie. Ak práve kreslíme, budeme si v premenných `oldx` a `oldy` pamätať súradnice bodu, do ktorého sme zatiaľ dokreslili. Ak sa myš počas kreslenia pohla, do premenných `newx` a `newy` si uložíme novú pozíciu myši, spojíme ju úsečkou so starou pozíciou a novú pozíciu si opäť zapamätáme ako `oldx` a `oldy`. Toto budeme robiť dovtedy, kým používateľ tlačidlo myši nepustí.

Samotný hlavný cyklus sa nachádza na riadkoch 18 až 58. Na začiatku skontrolujeme, či nenastalo ukončenie programu alebo či používateľ nestlačil `ESC` a ak áno, nastavíme `koncime` na 1 a opustíme cyklus.

Na riadkoch 32 až 39 spracujeme udalosť stlačenia tlačidla. Detaily o udalostiach typu `SDL_MOUSEBUTTONDOWN` a `SDL_MOUSEBUTTONUP` sú uskladnené v položke `event.button`. Ak chcete vedieť, ktoré tlačidlo myši bolo stlačené, použite podpoložku `event.button.button`. Ľavé tlačidlo je `SDL_BUTTON_LEFT`, pravé `SDL_BUTTON_RIGHT`, stredné `SDL_BUTTON_MIDDLE`. Ak potočíte kolieskom, tiež sa to knižnici oznámi ako udalosť stlačenia tlačidla. Pootočenie nahor vníma knižnica `SDL` ako stlačenie tlačidla `SDL_BUTTON_WHEELUP`, pootočenie nadol ako stlačenie tlačidla `SDL_BUTTON_WHEELDOWN`.

V prípade, že bolo stlačené ľavé tlačidlo, nastavíme hodnotu `kreslime` na 1 a do premenných `oldx` a `oldy` si uložíme pozíciu myši. Tá je uložená v položkách `event.button.x` a `event.button.x`.

Udalosť oznamujúcu, že sa myš pohla, spracúvame na riadkoch 40 až 49. Detaily o udalostiach typu `SDL_MOUSEMOTION` sú uložené v položke `event.motion`. Miesto, na ktoré sa myš pohla, je tradične uložené v položkách `event.motion.x` a `event.motion.y`. V položkách `event.motion.xrel` a `event.motion.yrel` sú uložené hodnoty o koľko sa

pozícia myši v jednotlivých smeroch zmenila od posledného pohybu. Tieto informácie sa môžu hodiť, ak programujete hru typu FPS kde pohyb myši určuje smer, ktorým sa postava pozerá. Vtedy totiž nepotrebujete určiť priamo súradnice kurzora myši, ale potrebujete zistiť, o koľko sa má zmeniť smer pohľadu postavy.

Na riadkoch 50 až 53 spracujeme udalosť zdvihnutia tlačidla myši. Pozrieme sa, či sa jednalo o ľavé tlačidlo myši a ak áno, nastavíme premennú `kreslime` na 0.

Na záver hlavného cyklu už len na riadku 57 prekreslíme okno.

Na záver ešte jedna funkcia, ktorá sa môže hodiť. Príkazom `SDL_ShowCursor(SDL_DISABLE);` môžete vypnúť kurzor myši. Ak namiesto `SDL_DISABLE` použijete `SDL_ENABLE`, myš sa opäť objaví.

Úloha 1: Pochopte a vyskúšajte.

Úloha 2: Zmeňte program tak, aby sa nakreslené zmazalo, keď stlačíte kláves C.

Úloha 3: Zmeňte program tak, aby ste pravým tlačidlom myši mohli kresliť žltou.

Úloha 4: Zmeňte program tak, aby sa na mieste, na ktoré kliknete stredným tlačidlom objavilo prasa zo štvrtej lekcie.

Úloha 5: Zmeňte program tak, aby sa po stlačení klávesy F nastavila kresleniu náhodná farba.

Úloha 6: Zmeňte program tak, aby sa pri stlačení šípky hore zväčšila hrúbka pera a pri stlačení šípky dole zmenšila hrúbka pera.

8. lekcia

Timer alebo „Presné ako hodinky“

V tejto lekcii sa budeme zaoberať problémom, ktorý sa vynoril kedysi dávno v počítačovom praveku, ale pri tvorbe počítačových hier straší dodnes. Totiž – herní programátori urobili hru, ľudia si ju kupovali a hrali a všetko bolo úžasné. Lenže človek (a ani firma) vývoj nezastaví. Počítače sú čím ďalej rýchlejšie. A keď niekto skúšal pustiť starú hru na novom výkonnom počítači, tak zrazu zistil, že sa všetko deje strašne rýchlo. Postavy sa rýchlo pohybujú, plošiny sa rýchlo uhýbajú, nepriatelia rýchlo tasia zbrane a hráč nestihne ani poriadne stláčať klávesy a už je mŕtvy.

Problém sa pôvodne riešil tak, že na počítač pridali tlačidlo TURBO, ktorým sa dala znížiť rýchlosť procesora. Lenže keď vznikli počítače, ktoré boli ešte rýchlejšie, prestalo byť únosné prepínať sa medzi toľkými rýchlosťami podľa toho, čo chce práve človek hrať.

A tak prišli ku slovu timery (po slovensky sa im niekedy hovorí aj časovače). Ide o to, že keď hra beží, pozerá sa pritom na hodinky. Keby sa veci mali diať príliš rýchlo, tak ich trochu pozdrží a keby niečo dôležité trvalo pridlho, tak nejaké menej dôležité veci vypustí.

Ukážeme si to na príklade. Začneme príkladom odstrašujúcim, ktorý v sebe žiadne timery nemá:

```
 1  #include <SDL/SDL.h>
 2  #include <SDL/SDL_gfxPrimitives.h>
 3  #include <time.h>
 4  #include <stdio.h>
 5
 6  #define abs(x) ((x)<0 ? -(x) : (x))
 7  #define min(x,y) ((x) < (y) ? (x) : (y))
 8
 9  typedef struct koliesko {
10    int x, y;  // poloha
11    int vx, vy; // rychlost
12    int rad;  // polomer
13    int r, g, b, a; // farba
14  } KOLIESKO;
15
16  void inicializujKoliesko(KOLIESKO *k, SDL_Surface *screen)
17  {
18    k->x = screen->w / 2;
19    k->y = 30;
20    k->vx = 0;
21    k->vy = 0;
22    k->rad = 20 + rand() % 10;
23    k->r = rand() % 256;
24    k->g = rand() % 256;
25    k->b = rand() % 256;
26    k->a = 255;
27  }
```

```
28  void kresli(KOLIESKO *k, SDL_Surface *screen, float medzifaza)
29  {
30    SDL_FillRect(screen,NULL,SDL_MapRGB(screen->format, 0, 0, 0));
31    filledCircleRGBA(screen,
32                     (int) (k->x + medzifaza * k->vx),
33                     (int) (k->y + medzifaza * k->vy),
34                     k->rad, k->r, k->g, k->b, k->a);
35    SDL_Flip(screen);
36  }
37
38  void pohniKoliesko(KOLIESKO *k, SDL_Surface *screen)
39  {
40
41    if (k->y > screen->h - k->rad)
42      k->vy = - abs(k->vy);
43    else
44      k->vy = k->vy + 1;
45    k->y = k->y + k-> vy;
46  }
47
48  int main(int argc, char *argv[])
49  {
50    SDL_Surface *screen = NULL;
51    KOLIESKO kol;
52    int koncime = 0;
53
54    SDL_Init( SDL_INIT_EVERYTHING );
55
56    screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE);
57    SDL_WM_SetCaption( "Ovladnute koliesko", NULL );
58    srand(time(NULL));
59
60    inicializujKoliesko(&kol,screen);
61
62    while (koncime == 0)
63    {
64      SDL_Event event;
65      while (SDL_PollEvent( &event ))
66      {
67        if (event.type == SDL_QUIT)
68           koncime = 1;
69        if (event.type == SDL_KEYDOWN &&
70                     event.key.keysym.sym == SDLK_ESCAPE)
71        {
72           koncime = 1;
73        }
74      }
75      pohniKoliesko(&kol, screen);
76
77      kresli(&kol, screen, 0);
78    }
79    SDL_Quit();
80    return 0;
81  }
```

Na riadkoch 1 až 4 klasicky includujme hlavičkové súbory. Súbor `time.h` potrebujeme iba kvôli generátoru náhodných čísel. Funkcie potrebné na prácu s časom má v sebe priamo knižnica SDL.

Na riadkoch 6 a 7 si definujeme makrá, ktoré nám budú vedieť nájsť absolútnu hodnotu čísla a minimum dvoch čísel. Makrá sú šikovná vec. Na prvý pohľad vyzerajú podobne, ako funkcie, ale fungujú trochu inak. Totiž ešte predtým, ako sa súbor skompiluje, prejde preprocesor celý váš súbor a všetky výskyty makra nahradí hodnotou, ktorá je uvedená vpravo od neho. To sa dá používať jednoduchým spôsobom, napríklad

```
#define EOF (-1)
```

Týmto spôsobom je hodnota `EOF`, ktorú poznáte z práce so súbormi skutočne definovaná v súbore `libio.h`, ktorý je načítaný zo súboru `stdio.h`. Preprocesor všade, kde uvidí `EOF` vloží miesto toho `(-1)`, ale ľuďom sa zdrojový kód, v ktorom majú to `EOF`, lepšie číta. Okrem toho ale môžu mať makrá aj parametre, čo sme využili v našom prípade.[20]

Na riadkoch 9 až 27 sme si urobili štruktúru `koliesko` a napísali sme funkciu, ktorá koliesko inicializuje. S touto štruktúrou ste sa už stretli, aj v tomto prípade bude fungovať tak, ako vždy.

Nasledujúce dve funkcie sa (samozrejme v trochu komplikovanejšej podobe) nachádzajú v každej hre. Funkcia `kresli` má na starosti vykresľovanie celej hernej scény. V našom prípade iba nakreslí koliesko. Všimnite si parameter `medzifaza`, ktorý slúži na to, aby ste mohli funkcii povedať, kde presne sa v pohybe nachádzate. Hodnota medzifázy by mala byť od 0 do 1. V prípade, že je 0, nakreslí sa koliesko na súradniciach `x` a `y`. V prípade, že je hodnota medzifázy 1, nakreslí sa koliesko na súradniciach `x+vx` a `y+vy`. Ak je hodnota medzifázy niečo medzi 0 a 1, nakreslí sa koliesko na správnom mieste medzi týmito dvoma bodmi.

Druhá funkcia `pohniKoliesko` reprezentuje hernú logiku. Táto funkcia má obyčajne na starosti načítanie vstupu od používateľa a s prihliadnutím na jeho akcie pomení polohu objektov na scéne, vyrieši prípadné úmrtia postáv a vypočíta všetko, čo bude kresliaca funkcia potrebovať, aby dobre kreslila. V našom prípade bude funkcia robiť iba to, že ak je koliesko na spodku obrazovky, obráti jeho rýchlosť, aby letelo zase hore, v opačnom prípade zväčší jeho rýchlosť smerom dole o 1, takže pohyb bude rovnomerne zrýchlený smerom nadol. Výsledný efekt by mal byť ten, že koliesko sa bude správať ako loptička – skákalka. Padne na spodok obrazovky, tam sa odrazí, vyletí opäť do pôvodnej výšky a takto bude skákať. A jeho pohyb bude dokonca fyzikálne správny.

Vo funkcii `main` okrem bežných vecí robíme iba to, že kým program neskončí, v cykle načítame a spracujeme udalosti, zavoláme funkciu `pohniKoliesko` a funkciu `kresli` (na medzifázu zatiaľ kašleme, je tam napevno nula).

Úloha 1: Pochopte, skompilujte a vyskúšajte.

---

20 Ak sa divíte, prečo sme do definície makra napchali toľko zátvoriek, objasníme to na nasledujúcom príklade. Predstavte si, že by sme mali makro definované nasledujúcim spôsobom:
`#define KRAT(x,y) x*y`
V programe by sme použili makro napríklad takto:
`a = KRAT(2+3,4+2);`
V premennej `a` by po tejto operácii napriek nášmu očakávaniu nebolo 30, ale 16. Totiž preprocesor to preloží do stavu
`a = 2+3*4+2;`
a keď to bude kompilátor vyhodnocovať, bude prihliadať na to, že násobenie má prioritu pred sčítaním, vypočíta najprv `3*4` a potom k tomu pripočíta tie dve dvojky. Aby sa takémuto chaosu predišlo, je rozumné používať aj okolo parametrov aj okolo výsledku celého makra zátvorky. Správna definícia by teda bola
`#define KRAT(x,y) ((x)*(y))`

Ak ste program spustili, problém, ktorý sme načrtli na začiatku lekcie sa vám ukázal v celej kráse. Ak nemáte práve predpotopný počítač, koliesko kmitá po obrazovke ako divé a nie je ho poriadne vidieť.

Prvé riešenie, ktoré predvedieme, je jednoduché. Ale ako to už s jednoduchými vecami býva, má svoje muchy. Tie ale rozoberieme až po ukážke. Zmeny sa týkajú iba funkcie main, všetko ostatné ostáva rovnaké, ako v predošlom príklade.

```
 1   int main(int argc, char *argv[])
 2   {
 3     SDL_Surface *screen = NULL;
 4     KOLIESKO kol;
 5     int koncime = 0;
 6     int posledne = 0;
 7     int TRVANIE_RAMCA = 1000 / 30;  /* Trvanie v milisekundach */
 8
 9     SDL_Init( SDL_INIT_EVERYTHING );
10
11     screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE);
12     SDL_WM_SetCaption( "Ovladnute koliesko", NULL );
13     srand(time(NULL));
14
15     inicializujKoliesko(&kol,screen);
16
17     while (koncime == 0)
18     {
19       if (SDL_GetTicks() - posledne > TRVANIE_RAMCA)
20       {
21         SDL_Event event;
22         while (SDL_PollEvent( &event ))
23         {
24             if (event.type == SDL_QUIT)
25               koncime = 1;
26             if (event.type == SDL_KEYDOWN &&
27                                 event.key.keysym.sym == SDLK_ESCAPE)
28             {
29               koncime = 1;
30             }
31         }
32         pohniKoliesko(&kol, screen);
33         posledne = SDL_GetTicks();
34       }
35
36       kresli(&kol, screen, (SDL_GetTicks() - posledne)/TRVANIE_RAMCA);
37     }
38     SDL_Quit();
39     return 0;
40   }
```

Kľúčom k časovaču je funkcia `SDL_GetTicks()`, ktorá vráti počet milisekúnd, ktoré uplynuli od inicializácie knižnice SDL a tým pádom funguje ako celkom presné hodinky.

Zavedieme si dve nové premenné. V premennej `posledne` si budeme pamätať, kedy sme naposledy prerátavali novú pozíciu kolieska. V premennej `TRVANIE_RAMCA` si budeme pamätať,

koľko milisekúnd nám má zabrať jeden snímok. Keď hodnotu nastavíme na `1000/30`, znamená to, že chceme približne tridsať snímkov za sekundu.

Celá finta spočíva v tom, že spracovanie udalostí a prepočítanie scény sa udeje len vtedy, ak už uplynul čas daného rámca. Celý blok kódu na riadkoch 19 až 34 sa vykoná iba vtedy, keď je splnená podmienka `SDL_GetTicks() - posledne > TRVANIE_RAMCA`. V prípade, že už dozrel čas na ďalší rámec, udalosti sa spracujú, koliesko sa posunie a hodnota premennej `posledne` sa nastaví na aktuálny čas (riadok 33).

Príjemné je aj to, že zatiaľ, čo dávame pozor na to, aby sa poloha neprepočítavala pričasto, nemusíme sa obmedzovať ohľadom toho, ako často koliesko vykresľujeme. Vypočítame si, v akej časti rámca sa práve nachádzame a výslednú hodnotu odovzdáme funkcii `kresli` (riadok 36). To nám umožňuje pri dostatočnom výkone procesora a grafickej karty vykresľovať pohyb ešte plynulejšie, ako jedenkrát počas rámca.

Úloha 2: Pochopte, skompilujte a vyskúšajte.

Momentálne to vyzerá tak, že svet je krásny, všetko funguje a problém je vyriešený. Tento pocit je ale daný tým, že naše funkcie sú úplne jednoduché. Ako by sa program správal, keby napríklad vykresľovacia funkcia zabrala viacej času, napríklad kvôli tomu, že ste program pustili na pomalšom hardvéri? To sa ľahko vyskúša. Stačí na koniec funkcie `kresli` pridať príkaz `SDL_Delay(200);` ktorý funkciu zdrží 200 milisekúnd.

Úloha 3: Vyskúšajte.

Funkcia, ktorá prepočítava polohu postáv býva väčšinou rýchlejšia, než tá, ktorá sa stará o vykresľovanie. Tá vykresľovacia funkcia je často aj závislejšia od hardvéru. A naše riešenie má presne tú slabinu, že keď bude táto funkcia zdržovať, nemáme to ako dobehnúť. Beh celého programu sa na pomalšom hardvéri neúnosne spomalí a tomu sme práve chceli predísť.

Preto predvedieme iné riešenie. Je to „riešenie na kľúč“ – také, ktoré je pokladané za to najlepšie, ktoré sa dá dosiahnuť. Pokojne ho preberte do svojich hier a upravte si ho tak, ako potrebujete. Zmeny sa opäť budú týkať iba funkcie `main`:

```
 1   int main(int argc, char *argv[])
 2   {
 3     SDL_Surface *screen = NULL;
 4     KOLIESKO kol;
 5     int koncime = 0;
 6     int posledne = 0;
 7     int aktualnyCas;
 8     int pocetCyklov;
 9     int TRVANIE_RAMCA = 1000 / 30;  /* Trvanie v milisekundach */
10     int MAX_POCET_CYKLOV = 10; /* Kolkokrat sa maximalne
11                                   moze pocitat bez kreslenia */
12     float percento;
13
14     SDL_Init( SDL_INIT_EVERYTHING );
15
16     screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE);
17     SDL_WM_SetCaption( "Ovladnute koliesko", NULL );
18     srand(time(NULL));
19
```

```
20      inicializujKoliesko(&kol,screen);
21
22
23      posledne = SDL_GetTicks();
24      while (koncime == 0)
25      {
26        aktualnyCas = SDL_GetTicks();
27        pocetCyklov = 0;
28        while (aktualnyCas - posledne > TRVANIE_RAMCA &&
29                                  pocetCyklov < MAX_POCET_CYKLOV)
30        {
31          SDL_Event event;
32          while (SDL_PollEvent( &event ))
33          {
34            if (event.type == SDL_QUIT)
35               koncime = 1;
36            if (event.type == SDL_KEYDOWN &&
37                       event.key.keysym.sym == SDLK_ESCAPE)
38            {
39               koncime = 1;
40            }
41          }
42
43          pohniKoliesko(&kol, screen);
44          posledne = posledne + TRVANIE_RAMCA;
45          pocetCyklov = pocetCyklov + 1;
46        }
47
48        if (aktualnyCas - posledne > TRVANIE_RAMCA) /* ! sietovky */
49          posledne = aktualnyCas - TRVANIE_RAMCA;
50
51        percento = min(1.f,
52                     ((float) (aktualnyCas - posledne))/TRVANIE_RAMCA);
53        kresli(&kol, screen, percento);
54      }
55      SDL_Quit();
56      return 0;
57    }
```

Oproti predošlému riešeniu si zriadime tri nové premenné. V premennej `aktualnyCas` budeme skladovať aktuálny čas na začiatku spracovania. Robíme to preto, lebo nemôžeme spoliehať na to, že sa počas prepočítavania novej polohy čas nezmení. (V minulom riešení sme na to spoliehali a to bola chyba.) Ďalšie dve premenné `pocetCyklov` a `MAX_POCET_CYKLOV` slúžia na nasledovnú vec: Ak by sa program príliš zdržal pri vykresľovaní, nebudeme novú polohu objektov na scéne počítať iba raz. Budeme to robiť až dovtedy, kým nedobehneme aktuálny čas. Mohlo by sa ale stať, že beh programu zaostáva za aktuálnym časom až príliš. A z času na čas niečo vykresliť treba. Preto v premennej `pocetCyklov` budeme počítať, koľkokrát už prepočet bežal a ak dosiahne hodnotu `MAX_POCET_CYKLOV`, tak prejdeme ďalej k vykresľovaniu. V prípade, že budete mať hodnoty nastavené tak, ako v ukážkovom programe, zaručí vám to vykreslenie približne trikrát za sekundu.

Hlavný cyklus hry sa nachádza na riadkoch 24 až 54. Na začiatku si zapamätáme aktuálny čas a počet prejdených cyklov nastavíme na nulu. Potom v ďalšom cykle na riadkoch 28 až 46

spracúvame používateľský vstup a hýbeme kolieskom až dovtedy, kým nedobehneme aktuálny čas (tak, že zakaždým zvýšime premennú `posledne` o trvanie jedného rámca) alebo kým celý cyklus neprejdeme maximálny povolený počet krát. Všimnite si, že vstupná podmienka cyklu je vymyslená tak, že ak ešte netreba nič prepočítavať, cyklus sa nevykoná ani raz.

Riadky 48 a 49 sú zaujímavé, pretože za niektorých okolností je vhodné ich použiť a za niektorých nie. Prídu ku slovu vtedy, keď cyklus, ktorý má na starosti výpočet novej polohy skončí skôr, ako stihne dobehnúť reálny čas. Ak programujete hru, ktorá beží na jednom počítači, tak je rozumné ten zvyšný čas zahodiť. Premennú `posledne` nastavíme len o jeden rámec za aktuálnym časom. To spôsobí, že hra sa trochu spomalí, ale pôjde plynulo a nebude sekať. Ak ale programujete sieťovú hru, pri ktorej je dôležitá časová synchronizácia medzi jednotlivými hráčmi, tak žiaden čas zahadzovať nesmiete. V tom prípade treba podstúpiť riziko, že hra môže niekomu sekať, pretože je dôležité, aby bežala všetkým rovnako rýchlo.

Na riadkoch 51 a 52 vypočítame, v akej časti snímku sa hra práve nachádza. Dávame pozor, aby výsledok bol maximálne 1, lebo ak by bola hodnota medzifázy väčšia, funkcia `kresli` by mohla robiť hlúposti.

A to je všetko. Ak počítač všetko stíha, nová pozícia sa počíta len vtedy, keď sa má a vykresľovanie sa volá tak často, ako sa dá. Ak prestane stíhať kreslenie, počítanie novej polohy sa spustí viackrát. Ak nestíha počítanie, tak je problém a hra naozaj bude behať pomalšie, ale to už sa nedá nič robiť, maximálne celú prepočítavaciu funkciu prerobiť, aby behala rýchlejšie.

Úloha 4: Pochopte, skompilujte a vyskúšajte.

Úloha 5: Spomaľte vykresľovanie (tak, ako v úlohe 3) a pozrite sa, čo to robí.

Úloha 6: Spomaľte prepočítavanie a pozrite sa, čo to robí.

Úloha 7: Pri spomalenom vykresľovaní skúste meniť hodnoty premenných `TRVANIE_RAMCA` a `MAX_POCET_CYKLOV` a pozerajte, čo to robí.

Úloha 8: Pri spomalenom prepočítavaní skúste meniť hodnoty premenných `TRVANIE_RAMCA` a `MAX_POCET_CYKLOV` a pozerajte, čo to robí.

9. lekcia

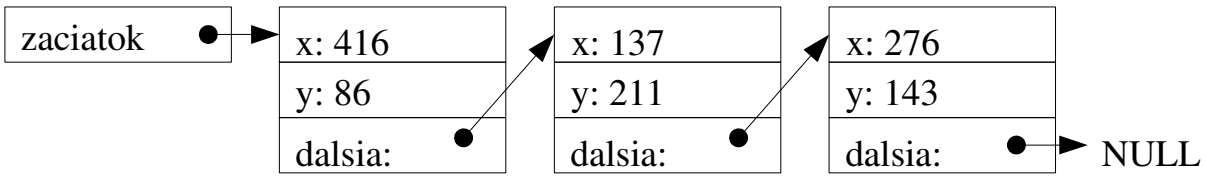
Zoznamy alebo „Ide vláčik ši, ši, ši“

Táto lekcia bude istým spôsobom výnimočná. Napriek tomu, že sa nachádza v cykle o knižnici SDL, nebudeme túto knižnicu tentokrát používať. Vystačíme si s obyčajným C-čkom. Totiž – v mnohých programoch je potrebné ukladať si informácie o rôznych objektoch do pamäte. Ak je počet objektov stále rovnaký, je možné vytvoriť si pole a do neho informácie ukladať. Ale je veľmi pravdepodobné, že počet ufónov, ktorí budú útočiť na vašu základňu sa bude meniť. A tu je vhodnejšie použiť iné dátové štruktúry, ako napríklad zoznamy.

Predstavte si, že si potrebujete uložiť údaje o viacerých potvorách, ktoré vo vašej hre budú útočiť na chudáka hráča. Pre jednoduchosť si budeme pamätať iba ich súradnice (v reálnej hre môže byť tých údajov oveľa viac). Najprv si vyrobíme novú štruktúru určenú pre jednu potvoru. Môže vyzerať napríklad takto:

```
typedef struct potvora
{
    int x;
    int y;
    struct potvora *dalsia;
} POTVORA;
```

Štruktúra `POTVORA` má tri zložky. Prvé dve – `x` a `y` budú obsahovať súradnice, na ktorých sa bude potvora nachádzať. Tretia zložka je smerník na ďalšiu štruktúru typu `POTVORA`. To nám umožní zoradiť všetky potvory ktoré sú práve na hracej ploche do vláčiku, ktorý bude vyzerať napríklad takto:

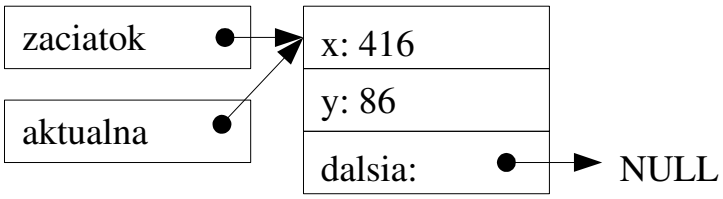


Dá sa to spraviť napríklad nasledujúcim spôsobom. Najprv si deklarujeme premennú `i` typu `int` a tri smerníky na štruktúru `POTVORA`, ktoré nazveme `zaciatok`, `aktualna` a `nova`. S pomocou funkcie `malloc` si v pamäti alokujeme prvú potvoru. Smerník `aktualna` nastavíme tak, aby na ňu ukazoval tiež. Hodnoty `x` a `y` nastavíme na náhodné hodnoty a smerník `dalsia` predbežne nastavíme na `NULL`.

```
int i;
POTVORA *zaciatok, *aktualna, *nova;

zaciatok = (POTVORA *) malloc(sizeof(POTVORA));
aktualna = zaciatok;
aktualna -> x = rand() % screen->w;
aktualna -> y = rand() % screen->h;
aktualna -> dalsia = NULL;
```

Keď tento kód prebehne, situácia bude približne takáto:



A teraz môžeme pridávať ďalšie vagóniky napríklad týmto spôsobom:

```
for( i = 0; i < 10; i++ )
{
    nova = (POTVORA *) malloc(sizeof(POTVORA));
    aktualna -> dalsia = nova;
    aktualna = nova;
    aktualna -> x = rand() % screen->w;
    aktualna -> y = rand() % screen->h;
    aktualna -> dalsia = NULL;
}
```

Smerník `aktualna` ukazuje na poslednú potvoru v zozname. V prvom príkaze v tele cyklu alokujeme v pamäti miesto pre novú potvoru. V druhom príkaze ju pripojíme na koniec vláčiku. Potom smerník `aktualna` nastavíme na novopridaný prvok a naplníme jednotlivé položky. V každom behu cyklu pridáme jeden vagónik. Keď ho teda necháme zbehnúť dvakrát, bude situácia vyzerať presne ako na prvom obrázku, ak prebehne desaťkrát, vagónikov bude 11.

Dôležité upozornenie: Ak neopatrne zmeníte hodnotu smerníka `zaciatok`, zabudnete, kde vláčik začína a ten začiatok sa nenávratne stratí v hlbinách pamäte počítača. Preto si dávajte pozor pri manipulácii s týmto smerníkom.

Ak chcete prejsť všetky prvky zoznamu, spraví sa to jednoducho. Nasledujúci cyklus vypíše súradnice všetkých zúčastnených potvor:

```
aktualna = zaciatok;
while (aktualna != NULL)
{
    printf("[%d, %d]\n", aktualna -> x, aktualna -> y);
    aktualna = aktualna -> dalsia;
}
```

Na začiatku smerník `aktualna` nastavíme na začiatok. Vypíšeme údaje potvory, na ktorú ukazuje a smerník `aktualna` nastavíme na ďalšiu. Toto opakujeme dovtedy, kým smerník `aktualna` ukazuje na niečo zmysluplné.

Keď prácu so zoznamom skončíme, patrí sa uvoľniť pamäť. Spravíme to napríklad takto:

```
while (zaciatok != NULL)
{
    aktualna = zaciatok -> dalsia;
    free((void *) zaciatok);
    zaciatok = aktualna;
}
```

Funguje to tak, že si v premennej `aktualna` uložíme druhý prvok v zozname, potom zmažeme začiatok zoznamu a smerník `zaciatok` nastavíme na nový začiatok vláčiku. Toto opakujeme, kým je čo mazať.

Uf. Je toho hodne na pochopenie. Takže si to pre istotu prečítajte ešte raz a potom sa pustite do úloh.

Úloha 1: Pochopte, poskladajte z toho zmysluplný program a vyskúšajte.

Úloha 2: Napíšte funkciu, ktorá v hotovom zozname vynuluje súradnice všetkých potvor.

Úloha 3: Napíšte funkciu, ktorá zo zoznamu vymaže piaty prvok, ak taký existuje. Vypíšte obsah zoznamu pred a po zavolaní funkcie.

Úloha 4: Vytvorte zoznam, v ktorom sa dá pohybovať oboma smermi. (Každý vagónik obsahuje okrem šípky na ďalšiu potvoru aj šípku na predošlú.) Vyrobte zoznam s desiatimi prvkami a vypíšte ho odpredu aj odzadu.

10. lekcia

Zvuk alebo „Randál musí byť“

Počítače majú už od čias počítačového praveku vstavané rôzne pípatká, reproduktorčeky a iné zariadenia schopné produkovať nejaký kravál. A rovnako oddávna programátori, ktorí robia hry, tieto zariadenia využívajú, aby počas víťazstva zneli víťazné fanfáry, aby počas úmrtia hráčovej postavičky znela patrične tragická muzika, aby sa počas pohybu po strašidelnom dome ozývali desivé zvuky v pozadí a aby sa počas veľkej kozmickej bitky ozývali z diaľky dunivé výbuchy. To, že sa zvuk vo vákuu nešíri a počas skutočnej kozmickej bitky by nebolo počuť nič, je úplne vedľajšie. Ide o atmosféru a o pôžitok z hry. Dobré ozvučenie je podstatnou zložkou dobrej hry a často zanechá väčší dojem, než vizuálna stránka.

Na to, aby ste mohli do svojej hry nejaké zvuky pridať, ich najprv musíte odniekiaľ získať. Sú dve možnosti, ako k zaujímavým zvukom prísť. Prvá možnosť je vytvoriť si ich. Stačí mikrofón a trocha fantázie – môžete nahrávať zvuk jablka padajúceho do umývadla, hrmenie v diaľke, bežiaci motor motorky alebo hluk staničnej haly. Ak ste muzikanti, máte kamarátov muzikantov alebo máte aspoň kúsok odvahy a softvér typu `garageband`, môžete si skúsiť spraviť do hry aj vlastnú hudbu.

Druhá možnosť je nájsť nejaké zvuky na internete. Existuje množstvo stránok, na ktorých môžete zadarmo a legálne získať hudbu a zvuky, ktoré môžete v hre použiť. Google je kamarát a pomôže vám nájsť, čo potrebujete. Za všetky tu spomeňme len dve stránky. Prvá je http://soundbible.com, z ktorej máme súbor `barreta_m9-Dion_Stapper-1010051237.wav` (výstrel z pištole a vypadnutie nábojnice, autor Dion Stapper) a ktorá obsahuje veľké množstvo zvukov, ktoré sú k dispozícii zadarmo pod rôznymi slobodnými licenciami. Druhá je stránka http://incompetech.com, ktorú vytvoril a spravuje Kevin MacLeod a na ktorej zverejnil mnoho svojich zaujímavých výtvorov pod licenciou Creative Commons. (Znamená to, že ich môžete používať zadarmo, ale musíte ho uvádzať ako autora.) Ak vám žiadna hudba z tej stránky nevyhovuje, môžete si Kevina najať a spraví vám hudbu na objednávku – živí sa tým. Stiahli sme od neho hororovú hudbu The House of Leaves[21] a pomocou programu `audacity` sme ju skonvertovali do formátu Ogg Vorbis (súbor `TheHouseOfLeaves.ogg`) lebo ten mp3 súbor knižnica SDL nejak nezvládla načítať.

Zvuky teda máme a radi by sme ich v našom programe použili. Opäť máme dve možnosti. Prvá je knižnica `SDL_sound`. Tá vám umožňuje prehrávať zvukové súbory v rôznych formátoch, ale pracuje sa s ňou dosť ťažkopádne. Preto je lepšie siahnuť po knižnici `SDL_mixer`, ktorej autorom je Jonathan C. Atkins.

Použitie si predvedieme na ukážkovom programe:

```
1   #include <SDL/SDL.h>
2   #include <SDL/SDL_mixer.h>
3   #include <stdio.h>
4
5   int main(int argc, char *argv[])
6   {
7     SDL_Surface *screen = NULL;
8     int koncime = 0;
```

21 The House of Leaves je veľmi zaujímavá knižka, ktorej autorom je Mark Z. Danielewski. Ak máte radi horory, komplikovanú literatúru a angličtinu, určite si ju zožeňte. http://en.wikipedia.org/wiki/House_of_Leaves

```
 9     Mix_Chunk *barreta = NULL;
10     Mix_Music *podmaz = NULL;
11
12     SDL_Init( SDL_INIT_EVERYTHING );
13
14     screen = SDL_SetVideoMode(640, 480, 32, SDL_SWSURFACE);
15     SDL_WM_SetCaption( "Zvuky", NULL );
16
17     SDL_FillRect(screen,NULL,SDL_MapRGB(screen->format, 0, 0, 0));
18
19     if (Mix_OpenAudio(22050, MIX_DEFAULT_FORMAT, 2, 1024) == -1)
20     {
21       printf("Mix_OpenAudio: %s\n", Mix_GetError());
22       exit(1);
23     }
24
25     barreta = Mix_LoadWAV("barreta_m9-Dion_Stapper-1010051237.wav");
26     podmaz = Mix_LoadMUS("TheHouseOfLeaves.ogg");
27     if (podmaz == NULL)
28     {
29       printf("Nejak to nenahralo... %s\n",Mix_GetError());
30       exit(1);
31     }
32
33     Mix_PlayMusic(podmaz, -1);
34
35     while (koncime == 0)
36     {
37       SDL_Event event;
38       while (SDL_PollEvent( &event ))
39       {
40         switch (event.type)
41         {
42             case SDL_QUIT:
43               koncime = 1;
44               break;
45             case SDL_KEYDOWN:
46               if (event.key.keysym.sym == SDLK_ESCAPE)
47                 koncime = 1;
48               break;
49             case SDL_MOUSEBUTTONUP:
50               if (event.button.button == SDL_BUTTON_LEFT)
51                 Mix_PlayChannel(-1, barreta, 0);
52               break;
53         }
54       }
55     }
56     Mix_FadeOutMusic(5000);
57
58     Mix_FreeMusic(podmaz);
59     Mix_FreeChunk(barreta);
60     SDL_Quit();
61     return 0;
62   }
```

Ako ste si mohli všimnúť, na riadku 2 nám medzi načítanými hlavičkovými súbormi pribudol súbor `SDL/SDL_mixer.h`.

Knižnica `SDL_mixer` pracuje s dvomi základnými dátovými štruktúrami. `Mix_Chunk` slúži na uloženie kratších zvukov a `Mix_Music` slúži na ukladanie hudby. Na riadkoch 9 a 10 sme deklarovali smerník na `Mix_Chunk` s názvom `barreta` a smerník na `Mix_Music` s názvom `podmaz`.

Ďalšia zaujímavá vec sa deje až na riadkoch 19 až 23. Funkcia `Mix_OpenAudio` tam štartuje celý zvukový systém. Má štyri parametre. Prvý je vzorkovacia frekvencia, s ktorou sa budú zvuky prehrávať. CD kvalita je 44100 Hz, pre účely väčšiny hier bohato stačí 22050 Hz. Ak nechcete uvádzať túto hodnotu číselne, môžete použiť namiesto čísla makro `MIX_DEFAULT_FREQUENCY`. Druhý parameter je výstupný formát. Môže byť 8 alebo 16 bitový a môže používať čísla so znamienkom alebo bez znamienka. Opäť môžete smelo použiť makro, tentokrát `MIX_DEFAULT_FORMAT`. Prípadné ďalšie detaily nájdete v manuáli. Tretí parameter je počet výstupných kanálov. 1 je mono, 2 je stereo. Neskôr bude ešte reč o kanáloch mixéra, s nimi tento parameter nemá nič spoločné. Posledný parameter hovorí, v akých veľkých kúskoch (v bajtoch) sa budú dáta posielať na výstup. Hodnota 1024 by tiež mala byť v naprostom poriadku.

Ak sa niečo pri štartovaní zvukového systému nepodarí, funkcia `Mix_OpenAudio` vráti hodnotu -1. Ak sa niečo pokazí, na riadku 21 vypíšeme, že čo. Kompletnú správu podá funkcia `Mix_GetError()` ktorá vracia reťazec s popisom chyby.

Na riadkoch 25 a 26 načítame zvuky zo súboru. Na načítanie do `Mix_Chunk` sme použili funkciu `Mix_LoadWAV`. Táto funkcia dokáže nahrať súbory vo formáte `wav`, `ogg`, `voc` a ešte niektorých ďalších a vráti smerník na načítanú štruktúru. Do `Mix_Music` sme načítali muziku pomocou funkcie `Mix_LoadMUS`. Táto funkcia by mala vedieť pracovať s formátmi `wav`, `ogg`, `mp3`, `flac` a `midi`, ale keď sme to skúšali s `mp3`, tak s tým boli problémy. Na riadkoch 27 až 31 môžete vidieť, ako sa overuje, či načítanie prebehlo v poriadku. Opäť tam používame už známu funkciu `Mix_GetError()`.

Na riadku 33 spúšťame hudbu funkciou `Mix_PlayMusic`. Prvý parameter je smerník na načítanú hudbu, druhý parameter udáva, koľkokrát sa má hudba zopakovať. Ak je hodnota -1, hudba sa opakuje, až kým sa nejako nevypne. Ak je hodnota 0, zahrá sa to raz.

Ďalšie prehrávanie sa uskutoční, keď kliknete ľavým tlačidlom myši. Na riadku 51 sa volá funkcia `Mix_PlayChannel`, ktorá prehrá výstrel z barrety. Táto funkcia má tri parametre. Prvý parameter určuje kanál, na ktorom sa má zvuk prehrať. Knižnica `SDL_mixer` podporuje takýchto kanálov osem (kanál pre hudbu funguje samostatne a do tohto počtu sa neráta). Keď chcete prehrávať dva zvuky naraz, treba si dať pozor, aby sa neprehrávali obidva na tom istom kanáli. Vtedy by totiž zvuk, ktorý púšťate neskôr iba nahradil zvuk, ktorý ste pustili skôr. Keď ich ale pustíte na rôznych kanáloch, mixér ich zmixuje, takže počuť obidva naraz. Hodnota `-1` hovorí, že sa zvuk má prehrať na prvom neobsadenom kanáli. Druhý parameter je smerník na zvuk, ktorý sa má prehrať. Tretí parameter určuje, koľkokrát sa má zvuk opakovať. Tentokrát ale 0 znamená jedno prehratie, 1 znamená dve prehratia, 2 znamená tri prehratia atď. Hodnota -1 opäť znamená, že sa zvuk bude opakovať, až kým ho niečo nezastaví.

Muzika hrá a pištoľ na kliknutie strieľa až kým používateľ program neukončí. Na riadku 56 vypneme hudbu. Funkcia `Mix_FadeOutMusic` bude hudbu stišovať počas doby určenej parametrom. Hodnota sa udáva v milisekundách, takže 5000 znamená päť sekúnd. Na riadkoch 58 a 59 sa načítané zvuky uvoľnia z pamäti a program sa skončí.

Úloha 1: Pochopte a vyskúšajte. Pri kompilovaní nezabudnite pridať medzi knižnice `SDL_mixer`.

Ku kanálu môžete pridať niektoré zaujímavé efekty. Napríklad streľba v predošlom príklade. Zatiaľ to vyzerá tak, že strieľate vy. V hre typu FPS[22] ale hráč obvykle nebýva tým jediným, kto strieľa. A pôsobilo by zvláštne, keby po hráčovi strieľala nejaká iná postava kdesi zďaleka sprava, ale streľba by znela rovnako, ako keď strieľa hráč. To z akého smeru a z akej vzdialenosti má streľba či iný zvuk zaznieť, sa dá ovplyvniť práve pomocou efektov.

Pre potreby ukážky si najprv pridáme generátor náhodných čísel. Medzi načítané hlavičkové súbory pridáme

```
#include <time.h>
```

a kdesi na začiatok inicializácie pridáme

```
srand(time(NULL));
```

nech nám fungujú náhodné čísla tak, ako majú. Teraz nahradíme riadok 51 z predošlej ukážky nasledujúcim kódom:

```
1   {
2     Mix_HaltChannel(1);
3     if (!Mix_SetPosition(1, rand() % 360, rand() % 256))
4       printf("Nefunguje efekt... %s\n",Mix_GetError());
5     Mix_PlayChannel(1, barreta, 0);
6   }
```

Streľbu teraz budeme prehrávať na kanáli 1. Keď dôjde k novému výstrelu, najprv zastavíme prehrávanie predošlého (ak nejaký ešte beží) pomocou funkcie `Mix_HaltChannel`. Ak by sme to nespravili, nový výstrel by síce zrušil predošlý, ale nevykonalo by sa nastavenie efektu. Samotný efekt nastavíme na riadku 3 funkciou `Mix_SetPosition`. Prvý parameter udáva, na ktorý kanál sa bude efekt vzťahovať. Druhý parameter udáva, z ktorého smeru zvuk zaznie. 0 znamená spredu, 90, sprava, 180 zozadu a 270 zľava. Tretí parameter udáva vzdialenosť, z ktorej zvuk zaznie. 0 znamená úplne blízko, 255 znamená úplne ďaleko. Riadky 3 a 4 ukazujú, ako zistiť, či sa efekt nastavil správne. Na riadku 5 už len na patričnom kanáli prehráme výstrel.

Knižnica `SDL_mixer` obsahuje viaceré zaujímavé funkcie, napríklad funkcie na ovládanie hlasitosti hudby, zmena pozície v prehrávanej hudbe alebo ďalšie efekty. Záujemcovia si ich môžu naštudovať na stránke s dokumentáciou: http://www.libsdl.org/projects/SDL_mixer/docs

Úloha 2: Vyskúšajte si nastavenie efektu.

Úloha 3: Doplňte do programu, aby sa po stláčaní klávesy `m` hudba postupne stišovala a po stláčaní klávesy `p` postupne zhlasňovala.

22 First person shooter alias strieľačka.

11. lekcia
Sieť alebo „Počítače sa bavia“

Ak má jednu počítačovú hru hrať naraz viacero ľudí, máte v podstate dve možnosti. Buď ju budú hrať na jednom počítači, alebo na viacerých, ktoré budú spojené cez sieť. A v tejto lekcii sa budeme venovať práve tej druhej možnosti – ako to spraviť, aby sa počítače medzi sebou dohodli a aby jeden počítač vedel, čo sa práve deje v druhom.

Aby sa takéto veci programovali ľahšie, existuje rozšírenie knižnice `SDL_net`. Jeho autorom je rovnako ako v prípade knižnice `SDL_mixer` Jonathan C. Atkins.

Základom komunikácie medzi dvoma počítačmi je takzvaný IP protokol[23]. Momentálne tento protokol existuje v dvoch verziách, verzii 4 a 6, knižnica `SDL_net` podporuje zatiaľ len verziu 4.

Veci fungujú tak, že keď sa chcú počítače medzi sebou dohodnúť, tak jeden počítač je server – je to počítač, ktorý počúva na sieti a čaká, kedy od neho bude niečo chcieť. Tie počítače, ktoré sa k nemu pripájajú, sú klienti. Využívajú to, že niekto čaká na ich podnet a keď sa s ním spoja, môžu si s ním vymieňať informácie.

Každý počítač pripojený do siete pomocou IP protokolu má číslo – takzvanú IP adresu. Keď sa chce klient k nejakému serveru pripojiť, musí toto číslo poznať alebo si ho aspoň vedieť zistiť. Okrem toho každý počítač môže poskytovať viacero služieb. Jeden počítač môže byť napríklad naraz server pre webové stránky, poštový server aj server obsluhujúci vašu hru. Každú z týchto vecí obsluhuje v počítači iný program. Aby v tom nebol neporiadok, okrem IP adresy treba ešte uviesť aj číslo portu[24], ktoré hovorí, o akú službu sa klient uchádza. Napríklad webserver štandardné beží na porte 80 a poštový server na porte 25. Číslo portu, na ktorom bude bežať váš server, si môžete vybrať. Treba ale dodržať dve podmienky: Číslo by nemalo byť menšie alebo rovné ako 1024. Porty s číslami do 1024 sú totiž privilegované a operačný systém k nim väčšinou len tak hocikoho nepustí. Ale aj keď budete používať číslo väčšie než 1024, treba sa pozrieť, či na vašej sieti niekto nepoužíva službu, ktorá toto číslo portu používa, aby ste nespôsobili chaos.[25]

Dosť bolo teoretických rečí, poďme sa pozrieť, ako sa to používa. Nasledujúci program stiahne stránku z nášho školského webu:

```
 1   #include <SDL/SDL.h>
 2   #include <SDL/SDL_net.h>
 3   #include <stdio.h>
 4   #include <string.h>
 5
 6   int main(int argc, char *argv[])
 7   {
 8     SDL_Init( SDL_INIT_EVERYTHING );
 9     SDLNet_Init();
10
```

23 IP je skratka z „Internet Protocol“.

24 Porty nie sú priamo súčasťou IP protokolu. Väčšina služieb ale využíva niektorú nadstavbu nad IP protokolom – najčastejšie TCP (Transmission Control Protocol) alebo UDP (User Datagram Protocol) a oba tieto protokoly porty používajú. Knižnica `SDL_net` podporuje obidva.

25 Ak chcete zistiť, aké služby patria k akým číslam portov, pozrite si stránku http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml alebo pod linuxom súbor /etc/services. Tým, že sú obsadené aj porty nad 1024 sa ale nemusíte veľmi znepokojovať.

```
11      IPaddress ip;
12      SDLNet_ResolveHost(&ip, "www.smnd.sk", 80);
13      const char* poziadavka =
14            "GET /main/ HTTP/1.1\nHost: www.smnd.sk\n\n";
15
16      TCPsocket server = SDLNet_TCP_Open(&ip);
17      if (server == NULL)
18      {
19        printf("Siet nebude... %s\n",SDLNet_GetError());
20        exit(-1);
21      }
22      SDLNet_TCP_Send(server,poziadavka,strlen(poziadavka)+1);
23
24      char data[1025];
25      int velkost;
26
27      while((velkost = SDLNet_TCP_Recv(server,data,1024)) != 0)
28      {
29        data[velkost] = '\0';
30        printf("%s",data);
31      }
32
33      SDLNet_TCP_Close(server);
34      SDLNet_Quit();
35      SDL_Quit();
36      return 0;
37    }
```

Na riadkoch 1 až 4 sa nič prekvapivé nedeje. Načítame hlavičkový súbor `SDL_net.h` a okrem neho aj `srting.h`, lebo budeme potrebovať funkciu `strlen`. Na riadku 9 inicializujeme sieťový podsystém.

Na riadku 11 deklarujeme premennú `ip` typu `IPaddress` do ktorej uložíme IP adresu servera. Na to ukladanie nám bude slúžiť funkcia `SDLNet_ResolveHost`, ktorá sa nachádza hneď na ďalšom riadku. Táto funkcia má tri parametre. Prvý je smerník na premennú `ip` do ktorej sa tá adresa uloží. Druhý je meno počítača, na ktorý sa budeme chcieť pripojiť. Príjemné je, že stačí zadať meno a netreba zadávať číselnú podobu adresy. Funkcia `SDLNet_ResolveHost` sa totiž spýta DNS servera[26], aká je číselná adresa počítača s daným menom a sama ho nastaví. Namiesto `"www.smnd.sk"` by ste ale mohli pokojne použiť `"193.87.13.210"`, čo je číselná adresa nášho servera. Tretí parameter je číslo portu. Keďže budeme chcieť stiahnuť webovú stránku, bude to 80.

Keď od servera niečo chceme, musíme mu najprv povedať, čo. Každá služba má svoj vlastný spôsob komunikácie. My ideme komunikovať HTTP protokolom. To znamená, že pošleme serveru nasledujúci text:

```
1   GET /main/ HTTP/1.1
2   Host: www.smnd.sk
3
```

Ten prázdny riadok na konci je dôležitý. Preto reťazec na riadku 14 končí `\n\n` a nie len `\n`. Hovoríme serveru, že chceme indexový súbor z adresára `/main/`, že používame HTTP protokol vo verzii 1.1 a že sa pripájame k počítaču s menom www.smnd.sk.

26 DNS server je počítač niekde na sieti, ktorý má na starosť prekladanie ľuďom zrozumiteľných mien počítačov na číselné adresy.

Všetko je pripravené, môžeme sa skúsiť pripojiť. Na riadku 16 vytvoríme pomocou funkcie `SDLNet_TCP_Open` TCP socket[27]. V tomto momente vznikne spojenie medzi našim klientom a serverom. Ak spojenie nie je možné nadviazať, funkcia `SDLNet_TCP_Open` vráti hodnotu `NULL`. V takom prípade podáme chybovú správu a program ukončíme. Na určenie toho, k akej chybe došlo, môžeme použiť funkciu `SDLNet_GetError`.

Ak je spojenie úspešne nadviazané, pošleme serveru našu požiadavku (riadok 22). Zavoláme funkciu `SDLNet_TCP_Send`. Prvý parameter bude smerník na socket, druhý bude reťazec, v ktorom máme našu požiadavku uloženú a tretí je počet bajtov, ktoré majú byť odoslané. K dĺžke reťazca pripočítavame jednotku kvôli tomu, aby sa odoslal aj špeciálny znak `\0`, ktorým je každý reťazec ukončený.

Požiadavku sme odoslali. Teraz treba prečítať to, čo nám server pošle naspäť. Na riadku 24 si deklarujeme reťazec `data`, do ktorého budeme údaje od servera ukladať. Na čítanie zo servera slúži funkcia `SDLNet_TCP_Recv`. Má v podstate rovnaké parametre, ako `SDLNet_TCP_Send`. Prvý je smerník na socket, druhý je reťazec, do ktorého sa bude ukladať výstup a tretí je maximálny počet načítaných bajtov.[28] Funkcia vráti počet načítaných bajtov. Opakovane ju voláme (riadok 27), kým počet načítaných bajtov nie je 0 – vtedy už sme prečítali všetko. Na koniec načítaného úseku vždy pridáme znak `\0` (riadok 29) aby bol reťazec správne ukončený – kvôli tomu sme deklarovali reťazec dĺžky 1025 a znakov načítavame iba 1024. Načítaný úsek vypíšeme.

Keď je už všetko vypísané, zavrieme socket funkciou `SDLNet_TCP_Close` (riadok 33), ukončíme sieťovanie, knižnicu SDL a celý program.

Keď program spustíte, ukáže sa vám HTML kód našej titulnej stránky. Úplne rovnako si ho sťahujú prehliadače, tie ale ešte navyše vedia stránku zobraziť pekným písmom a pridať farbičky a obrázky.

Úloha 1: Pochopte a vyskúšajte.

Uf. Takže takto to zhruba funguje. Poďme si teraz ukázať, ako si napísať vlastný server. Kým sa ale pustíme do programovania, treba porozmýšľať, čo ten server má vlastne robiť.

Náš server bude úplne jednoduchý. Bude si pamätať súradnice maximálne dvadsiatich hráčov, bude ich vedieť meniť a na požiadanie klienta mu ich prezradí, bude vedieť hráčov pridávať a odoberať a bude sa dať klientom vypnúť. Server bude rozumieť nasledujúcim príkazom:

```
N
```

Server sa pokúsi vytvoriť nového hráča. Ak sa mu to podarí, vráti reťazec so znakom + a číslom vytvoreného hráča (číslo bude od 0 do 19), takže napr. „`+ 7`“, ak sa mu to nepodarí (napríklad preto, lebo všetci dvadsiati hráči sú už vytvorení), vráti reťazec „−“.

```
W 3 143 623
```

Server sa pokúsi nastaviť hráčovi s číslom 3 súradnice 143, 623. Ak bude pokus úspešný (číslo hráča nie je väčšie ako 19 ani menšie, ako 0, hráč s daným číslom existuje a obe súradnice majú hodnotu od 0 do 9999) súradnice sa nastavia a server vráti reťazec „+“. Inak vráti reťazec „−“.

```
R 3
```

Ak je uvedený index v rozsahu od 0 do 19 a hráč s daným číslom existuje, server vráti jeho súradnice v tvare „`+ 143 623`“. Inak vráti reťazec „−“.

---

27 Bežný preklad je „zástrčka“, významovo presnejšie by bolo „komunikačný kanál“

28 Dajte si pozor, aby ste nenačítali viac dát, než sa vám vmestí do reťazca. Chyba sa volá buffer overflow a šikovný hacker vie takúto chybu využiť k prieniku do počítača. http://en.wikipedia.org/wiki/Buffer_overflow

`K 3`

Ak je uvedený index v rozsahu od 0 do 19 a hráč s daným číslom existuje, server hráča zruší a vráti reťazec „+“. Inak vráti reťazec „−“.

`Q`

Server vráti reťazec „`+ 1337`“ a ukončí činnosť.

Vo všetkých ostatných prípadoch server vráti reťazec „−“.

Server, ktorý bude uvedené veci robiť, môže vyzerať napríklad takto:

```
 1  #include <SDL/SDL.h>
 2  #include <SDL/SDL_net.h>
 3  #include <stdio.h>
 4  #include <string.h>
 5
 6  int main(int argc, char *argv[])
 7  {
 8    int x[20], y[20];
 9    int n, index, xi, yi;
10    char operacia;
11    int bezime = 1;
12
13    SDL_Init( SDL_INIT_EVERYTHING );
14    SDLNet_Init();
15    for( n= 0; n < 20; n++ )
16      x[n] = -9999;
17
18    IPaddress ip;
19    SDLNet_ResolveHost(&ip, NULL, 4247);
20
21    TCPsocket server = SDLNet_TCP_Open(&ip);
22
23    while (bezime)
24    {
25      TCPsocket klient = SDLNet_TCP_Accept(server);
26      if (klient)
27      {
28        char poziadavka[256];
29        char odpoved[256];
30
31        SDLNet_TCP_Recv(klient,poziadavka,256);
32        n = sscanf(poziadavka,"%c %d %d %d",
33                               &operacia, &index, &xi, &yi);
34        printf("%d %c %d %d %d\n", n, operacia, index, xi, yi);
35
36        if (n == 4 && operacia == 'W')
37        {
38           if (xi >= 0 && xi <= 9999 && yi >= 0 && yi <= 9999
39                       && index >= 0 && index < 20
40                       && x[index] > -9999)
41           {
42             x[index] = xi;
43             y[index] = yi;
44             sprintf(odpoved,"+\n");
45           }
```

```
46            else
47              sprintf(odpoved,"-\n");
48          }
49          else if (n == 2 && operacia == 'R')
50          {
51             if (index >= 0 && index < 20 && x[index] > -9999)
52               sprintf(odpoved,"+ %d %d\n",x[index],y[index]);
53             else
54               sprintf(odpoved,"-\n");
55          }
56          else if (n == 2 && operacia == 'K')
57          {
58             if (index >= 0 && index < 20 && x[index] > -9999)
59             {
60               x[index] = -9999;
61               sprintf(odpoved,"+\n");
62             }
63             else
64               sprintf(odpoved,"-\n");
65          }
66          else if (n == 1 && operacia == 'N')
67          {
68             int i;
69             for( i = 0; i < 20; i++)
70               if (x[i] == -9999)
71                 break;
72             if (i < 20)
73             {
74               x[i] = 0;
75               y[i] = 0;
76               sprintf(odpoved,"+ %d\n",i);
77             }
78             else
79               sprintf(odpoved,"-\n");
80          }
81          else if (n == 1 && operacia == 'Q')
82          {
83             sprintf(odpoved,"+ 1337\n");
84             bezime = 0;
85          }
86          else
87             sprintf(odpoved,"-\n");
88          SDLNet_TCP_Send(klient,odpoved,strlen(odpoved)+1);
89          SDLNet_TCP_Close(klient);
90          for(n = 0; n < 256; n++)
91             poziadavka[n] = '\0';
92        }
93      }
94
95      SDLNet_TCP_Close(server);
96      SDLNet_Quit();
97      SDL_Quit();
98      return 0;
99    }
```

Na riadkoch 8 až 11 deklarujeme premenné. Máme tam dvadsaťprvkové polia `x` a `y` do ktorých budeme ukladať súradnice, premenné `operacia`, `index`, `xi` a `yi` do ktorých si budeme vedieť uložiť príkaz pre server, premennú `n`, do ktorej budeme ukladať, koľko položiek príkaz pre server mal a premennú `bezime`, ktorá bude určovať, či má hlavný cyklus stále bežať, alebo či sa už má skončiť.

Na riadkoch 13 až 16 všetko zinicializujeme. Všetky položky poľa `x` nastavíme na hodnotu `-9999`. Táto hodnota bude symbolizovať, že hráč nie je aktívny.

Na riadku 19 nastavujeme premennú `ip` pre server. To, že pôjde o server sa spozná podľa toho, že druhý parameter neobsahuje meno ani adresu iného počítača, ale `NULL`. Číslo portu bude 4247. Server bude teda počúvať na tomto porte, či sa naň niekto nepripojí. Na riadku 21 vytvoríme socket pre server (od tohto momentu server začne počúvať) a môžeme začať spracovávať požiadavky.

Hlavný cyklus sa nachádza na riadkoch 23 až 93. Na riadku 25 voláme funkciu `SDLNet_TCP_Accept(server)`. Táto funkcia pozrie, či sa na server niekto pripája. Ak nie, vráti `NULL`, ak áno, vráti smerník na socket k novému klientovi. Na riadku 26 sa teda pozrieme, či je niekto pripojený (hodnota socketu `klient` nie je `NULL`) a ak je pripojený, môžeme sa s ním začať baviť.

Deklarujeme si dva reťazce, `poziadavka` a `odpoved`. Na riadku 31 sa klienta opýtame, čo od nás vlastne chce a jeho odpoveď uložíme do reťazca `poziadavka`.

Reťazec `poziadavka` teraz musíme prečítať. Použijeme na to funkciu `sscanf`, ktorá funguje rovnako, ako funkcia `scanf`, ale nečíta veci zo štandardného vstupu, ale z reťazca. Prvý parameter je reťazec, z ktorého sa bude čítať, druhý je formátovací reťazec vstupu a potom nasledujú smerníky na premenné, do ktorých sa hodnoty majú uložiť. Keďže všetky príkazy pre server vyzerajú tak, že na začiatku je písmeno a potom nasledujú nula až tri čísla, formátovací reťazec bude `"%c %d %d %d"`. Funkcia `sscanf` má okrem iného tú výhodu, že vráti počet úspešne načítaných premenných. (Mimochodom, túto vlastnosť má aj funkcia `scanf`.) Ak teda požiadavka na server bude „`N`“, premenná `n` bude mať hodnotu 1, lebo sa načíta iba jedna premenná. Ak bude požiadavka „`R 17`“ `n` bude mať hodnotu 2 a ak bude požiadavka „`W 5 314 271`“, `n` bude mať hodnotu 4. Server na riadku 34 vypíše, ako sa mu podarilo požiadavku prečítať. Je to iba kontrolný výpis pre našu informáciu.

Keď už vieme, čo od nás klient chce, môžeme sa začať venovať spracovávaniu požiadavky. Ak chce, aby sme zapísali súradnice hráča (písmenko `W` a štyri parametre, riadky 36 až 48), tak ich zapíšeme. Do reťazca `odpoved` vložíme odpoveď pre klienta. Používame na to funkciu `sprintf`, ktorá funguje rovnako ako `printf`, akurát svoj výstup ukladá do reťazca. Na riadkoch 49 až 55 klientovi vrátime informácie o súradniciach hráča (písmenko `R`, 2 parametre), na riadkoch 56 až 65 hráča zmažeme (písmenko `K`, 2 parametre). Na riadkoch 66 až 80 nájdeme voľnú pozíciu a vrátime ju hráčovi (písmenko `N`, 1 parameter). Všimnite si spôsob, ktorým voľnú pozíciu hľadáme (riadky 69 až 71). Použili sme `for` cyklus, v ktorom premenná `i` prechádza všetky hodnoty od 0 do 19. Ak nájdeme prázdnu pozíciu, prerušíme cyklus príkazom `break`, takže premenná `i` bude mať hodnotu voľnej pozície. Ak bude hodnota `i` 20, znamená to, že za žiadna voľná pozícia nenašla a cyklus skončil vďaka porušeniu vstupnej podmienky. Na riadkoch 81 až 85 spracujeme zastavenie servera. V prípade, že žiadna z možností nenastala, na riadku 87 nastavíme negatívnu odpoveď.

Na riadku 88 pošleme klientovi odpoveď a na riadku 89 spojenie s ním ukončíme. Na riadkoch 90 až 91 vynulujeme reťazec s požiadavkou, nech sa staré hodnoty nemiešajú do nových požiadaviek.

Keď hlavný cyklus skončí, zavrieme socket servera a všetko patrične ukončíme.

Úloha 2: Pochopte a skompilujte.

Prv, než napíšeme program klienta, ktorý by s týmto serverom vedel spolupracovať, môžeme náš server vyskúšať pomocou programu `telnet`. Pod linuxom by mal byť štandardne prístupný. Verzia pre windows existuje tiež, ale keďže posiela serveru požiadavky po každom stlačení klávesu namiesto po každom stlačení enteru, je pre naše potreby nepoužiteľná. Ak chcete teda v linuxe zadať serveru príkaz, najprv spustite server, potom otvorte ďalšiu konzolu a do príkazového riadku napíšte

```
telnet 127.0.0.1 4247
```

Adresa `127.0.0.1` znamená „tento počítač“, druhý parameter je číslo portu. Program sa rovno spojí na server a do príkazového riadku môžete napísať príkaz pre server. Program `telnet` vypíše, čo server odpovedal a skončí, lebo server ukončil spojenie. Môžete stlačiť šípku hore, program spustiť znovu a zadať mu ďalší príkaz.

So skúsenosťami, ktoré máte, by ste dokázali klienta k tomuto serveru napísať aj sami. Pozrite sa ale na toho, ktorého sme vytvorili my:

```
 1  #include <SDL/SDL.h>
 2  #include <SDL/SDL_net.h>
 3  #include <stdio.h>
 4  #include <string.h>
 5
 6  IPaddress ip;
 7
 8  int poziadavkaServer(const char* poziadavka, int* h1, int* h2)
 9  {
10    char odpoved[256], navrat;
11    int a, b, n;
12
13    for(a = 0; a < 256; a++)
14      odpoved[a] = '\0';
15    TCPsocket server = SDLNet_TCP_Open(&ip);
16    if (server == NULL)
17    {
18      printf("Socket nebude... %s\n",SDLNet_GetError());
19      return 0;
20    }
21    SDLNet_TCP_Send(server,poziadavka,strlen(poziadavka)+1);
22    SDLNet_TCP_Recv(server, odpoved, 256);
23    if (odpoved[0] != '+')
24      return 0;
25    n = sscanf(odpoved, "%c %d %d", &navrat, &a, &b);
26    if (n >= 2)
27      *h1 = a;
28    if (n == 3)
29      *h2 = b;
30    SDLNet_TCP_Close(server);
31    return 1;
32  }
33
34  int main(int argc, char *argv[])
35  {
36    int a,b,ret;
37
```

```
38      SDL_Init( SDL_INIT_EVERYTHING );
39      SDLNet_Init();
40
41      SDLNet_ResolveHost(&ip, "127.0.0.1", 4247);
42
43      ret = poziadavkaServer("N", &a, &b);
44      printf("%d %d\n", ret, a);
45      ret = poziadavkaServer("W 1 42 47", &a, &b);
46      printf("%d\n", ret);
47      ret = poziadavkaServer("W 0 42 47", &a, &b);
48      printf("%d\n", ret);
49      ret = poziadavkaServer("R 0", &a, &b);
50      printf("%d %d %d\n", ret, a, b);
51      ret = poziadavkaServer("R 1", &a, &b);
52      printf("%d\n", ret);
53      ret = poziadavkaServer("K 0", &a, &b);
54      printf("%d\n", ret);
55      ret = poziadavkaServer("R 0", &a, &b);
56      printf("%d\n", ret);
57      ret = poziadavkaServer("Q", &a, &b);
58      printf("%d %d\n", ret, a);
59
60      SDLNet_Quit();
61      SDL_Quit();
62      return 0;
63  }
```

Na riadkoch 8 až 32 máme funkciu `poziadavkaServer`, ktorá má tri parametre. Prvý je reťazec s príkazom pre server, ďalšie dva sú smerníky na celé čísla, ktoré sa naplnia odpoveďou zo servera (iba v prípade príkazov `R` a `Q`[29]). Ak odpoveď nezačína na `'+'`, rovno vrátime ako hodnotu 0 – neúspech. Na čítanie odpovede od servera používame príkaz `sscanf` rovnakým spôsobom, ako sme ho použili pri serveri. Na riadkoch 43 až 58 posielame serveru rôzne príkazy, necháme vypisovať návratové hodnoty a kocháme sa, ako to pekne funguje.

Úloha 3: Pochopte a vyskúšajte.

Úloha 4: Predstavte si, že okrem hráčov sa na hracom pláne ešte pohybujú aj nejaké strely. Čo myslíte? Ktorá strana programu by mala riešiť kolízie a prípadné úmrtia? Klient alebo server? Prečo?

Úloha 5: Dorobte do servera príkaz „`D 3`“, ktorý má ako ďalší parameter číslo hráča a ak je index v poriadku a taký hráč existuje, server mu náhodne zmení pozíciu (ale tak, aby boli súradnice stále v intervale 0 až 9999). Ak index v poriadku nie je, server vráti informáciu o neúspechu. Dorobte testovanie tohto príkazu do klienta.

Úloha 6: Vedeli by ste prerobiť server tak, aby nebol obmedzený počet hráčov? Čo všetko by bolo treba urobiť?

29 A to ešte v prípade príkazu `Q` návratová hodnota nemá žiaden význam a je tam len na okrasu.

12. lekcia

Výzva alebo „Čo teraz?“

No čo. Naprogramujte niečo veľkolepé.

Úloha 1: Urobte to.